**MASTER EN SCIENCES INFORMATIQUES**

# Construction d'un environnement d'aide à la programmation d'un micro-robot

Vanhemelryck, Claude

*Award date:*
1985

*Awarding institution:*
Universite de Namur

# CONSTRUCTION D'UN ENVIRONNEMENT D'AIDE A LA PROGRAMMATION D'UN MICRO-ROBOT

MEMOIRE PRESENTE PAR

CLAUDE VANHEMELRYCK

EN VUE DE L'OBTENTION DU TITRE

DE LICENCIE ET MAITRE EN INFORMATIQUE

Année Académique 1984/1985

Je tiens à remercier Monsieur Axel VANLAMSWEERDE, promoteur de ce mémoire, ainsi que Monsieur Luc DEKEYSER, mon maître de stage pour l'aide inestimable qu'ils m'ont apporté durant l'élaboration de ce travail.

Je tiens à remercier tous les membres du personnel du service d'informatique médicale de l'Hôpital Gasthuiberg à Louvain pour leur chaleureux accueil et leur grande compréhension.

Je tiens à remercier Monsieur Stan WIJNS pour avoir mis un robot à mon entière disposition.

Enfin, je remercie Mademoiselle Anne LEMOINE et Madame Christiane BUELENS qui ont bien voulu dactylographier le présent travail.

TABLE DES MATIERES
==========



...

...

...

...

# 1. INTRODUCTION

Le robot est devenu un mythe. On le présente comme un être mécanique, constiuté d'organes mystérieux, capable d'agir comme un homme, doué de la parole, mettant une force surhumaine au service d'une intelligence primitive et animé de sinistres intentions.

Si le mythe du robot date de plusieurs siècles, le mot "robot" n'est apparu que très récemment. Il trouve en effet son origine dans une pièce de l'écrivain tchèque Karel CAPEK, écrite en 1923 et intitulée R.U.R. ou les robots universels de Rossum.

Le mot "robota" en Tchèque signifie travail ou corvée. Les robots de Capek sont des humanoïdes créés par Rossum et son fils, car ils voyaient en eux le moyen de débarasser l'homme des tâches pénibles. Mais l'histoire finit bien mal : les robots finissent par se rebeller et détruire l'humanité.

Le mythe du robot entretenu par les ouvrages de science-fiction n'a certainement pas favorisé son insertion dans le processus de production. Par ailleurs, on voit en lui un danger pour l'emploi et pour la qualité de l'emploi. L'ère de la robotique sera peut-être également celle de la déqualification professionnelle d'une masse de travailleurs manuels, reléguée à une tâche de surveillance du processus de production.

La réalité de la robotique est heureusement moins sombre et certainement fort éloignée du mythe des monstres antropomorphiques. La robotique a connu son véritable démarrage au cours des années 70. Depuis, elle fait l'objet d'énormes efforts de recherche. La robotique répond au besoin d'automatisation de l'industrie occidentale et c'est pour cette raison qu'elle se développe aussi rapidement. L'intérêt du robot par rapport aux autres outils automatiques provient de son aptitude a être (re)programmé, sa souplesse et sa précision. Il permet aussi d'accomplir des tâches qui étaient irréalisables jusqu'à présent à cause du danger qu'elles comportaient, ou à cause des limites physiques de l'être humain. Citons , par exemple, l'exploration des hauts fonds sous-marins, l'entretien des parois internes des oléoducs ou encore la manipulation des produits radio-actifs. C'est certainement pour cette raison que l'on désigne une application robotique par le vocable d'"atelier flexible".

Nous allons passer en revue quelques pôles de la robotique afin de bien comprendre ce qu'est un robot aujourd'hui, ce qu'il est capable de faire et surtout voir comment on peut le programmer.

- Les différents types de robots

(Dossier Athéna n° 1 - 1er avril 1984)

Il existe à l'heure actuelle trois catégories de robots sur le marché :

- Les robots de production : on les appelle aussi les robots industriels. Ils sont destinés à accomplir des tâches répétitives dans un environnement quasi invariant. L'application la plus spectaculaire est certes, l'industrie de l'automobile. Les exemples typiques d'utilisation sont la soudure, la peinture, la manutention, le tri de pièces, l'assemblage et le contrôle de qualité.
- Les robots d'exploration : ce type de robot a peu de chose en commun avec le type précédent. La problématique de ce type de robot tient au fait que la machine connaît son objectif mais que son univers n'est plus invariant. Pour résoudre ce problème, il y a deux approches.
  - soit on ajoute un agent humain dans la boucle de la commande et on parlera de téléopération;
  - soit on dote la machine de capteurs evolués et d'algorithmes de traitement et d'interprétation des informations percues. On parlera alors de robots "intelligents" ou autonomes.
- Les robots d'assistance : ce sont des prothèses sophistiquées visant à rendre une certaine autonomie à un être humain handicapé.

Dans le cadre de ce travail, nous ne nous intéresserons qu'au premier type de robot.

- Les générations des robots

- La première génération : si l'on ne tient pas compte des prototypes particulièrement performants permettant aux grands laboratoires de recherche de travailler sur des projets de

...

robotique avancés, la grande majorité des robots sont de la première génération. Ce genre de robot est démuni de capteurs sensoriels et est destiné à accomplir des tâches répétitives dans un environnement sans surprise. On distingue deux types de robot de la première génération :

- le robot "playback". Les opérations à exécuter sont mémorisées en exécutant une première fois l'opération manuellement. Ensuite, elles peuvent être répétées automatiquement.
- le robot à contrôle numérique : exécute une série d'informations d'après une série de commandes chargées numériquement. Ces commandes définissent entre-autre, la succession des positions du robot.

- La deuxième génération : les robots de la deuxième génération sont des robot à contrôle numérique dotés de fonctions de perception avancées. L'informatique joue un rôle prépondérant dans la commande de ces robots. Un ordinateur sera couplé au robot. Sa fonction consistera à traiter ces données et d'assurer le contrôle et la coordination des mouvements du robot. Cette génération s'ammorce à peine et reste encore pour l'essentiel, du domaine des laboratoires industriels ou universitaires.
- La troisième génération : les robots de la troisième génération sont caractérisés par une grande autonomie de décision. Ils sont capables d'évoluer dans un environnement imprévisible. Ils doivent donc être capables de choisir et d'entreprendre les actions à accomplir d'après les informations fournies par l'environnement et fournies à l'aide de senseurs. Ici, il n'y a donc plus d'ordre prédéterminé des actions. A l'heure actuelle, la troisième génération est encore à l'état de projet.

- <u>Les langages de programmation</u>

Les systèmes robotiques à commande numérique disposent d'un langage de programmation permettant de spécifier les mouvements à accomplir. Les langages de programmation des robots de la deuxième génération comprennent des commandes spécifiques supplémentaires permettant l'utilisation des capteurs sensoriels.

...

On qualifie ces langages "d'explicite" ou encore "robot-level".

Grâce à l'utilisation des informations délivrées par les capteurs sensoriels, le robot à désormais la possibilité de travailler dans un environnement plus incertain. Ce qui a pour conséquence d'élargir considérablement leur champ d'application. Le gros désavantage de ces langages, c'est qu'ils sont directement liés aux caractéristiques physiques du robot et des capteurs. Aussi n'est-il pas étonnant qu'il y ait presque autant de langage de programmation qu'il y a de constructeurs de robot. L'utilisation de ces langages requiert une expertise du programmeur en informatique et demande une grande habileté pour le choix des mouvements spécifiquement liés à l'utilisation des capteurs sensoriels.

Il existe une autre approche de la programmation des robots. Elle est connue sous le nom de programmation du niveau "tâche" (task-level). La programmation consiste uniquement à spécifier les actions à entreprendre (position finale des objets) sans spécifier les mouvements que doit accomplir le robot en vue de ces actions. Cette programmation est donc complètement indépendante des caractéristiques physiques du robot. Bien sûr, les systèmes de programmation "task-level" utilisent une modélisation géométrique de l'espace de travail du robot et du robot lui-même. La programmation "task-level" est encore un sujet de recherche et n'est donc pas encore utilisée dans l'industrie.

# 2. CADRE GENERAL

## 2.1. DEFINITIONS ET COMPOSITION DU ROBOT INDUSTRIEL

### 2.1.1. Définitions

(Extraites de la norme française NF E 61-100 août 1983)

Manipulateur : Mécanisme généralement composé d'éléments en série, articulés ou coulissant l'un par rapport à l'autre, dont le but est la saisie et le déplacement d'objets suivant plusieurs degrés de liberté. Il est multifonctionnel et peut être commandé directement par un opérateur humain ou par tout système logique (système à cames, logique pneumatique, logique électrique cablée ou programmée).

Robot industriel : Manipulateur automatique, asservi en position, capable de positionner et d'orienter des matériaux, pièces, outils ou dispositifs spécialisés, au cours de mouvements variables et programmés pour l'exécution de tâches variées.
Il se présente souvent sous la forme d'un ou plusieurs bras se terminant par un poignet. Son unité de commande utilise, notamment, un dispositif de mémoire et éventuellement de perception de l'environnement et des circonstances ainsi que l'adaptation en résultant.
Ces machines polyvalentes sont généralement étudiées pour effectuer la même fonction cyclique et peuvent être adaptées à d'autres fonctions sans modification permanente du matériel.

Degré de liberté : Un mode, suivant lequel un système dynamique peut être modifié. Chacun de ces modes est caractérisé par une variable indépendante. Le nombre de degrés de liberté d'un robot industriel est limité; il est lié à son aptitude à positionner et orienter son organe terminal; ainsi sa valeur théorique maximale est de 6.

2.1.2. Composition du robot industriel

Comme nous l'avons vu dans la définition, le robot industriel est constitué de plusieurs éléments. Ceux-ci travaillent en collaboration afin de permettre à chaque membre du bras articulé de pouvoir prendre une séquence de positions pour une application donnée.

Ces éléments sont les suivants :

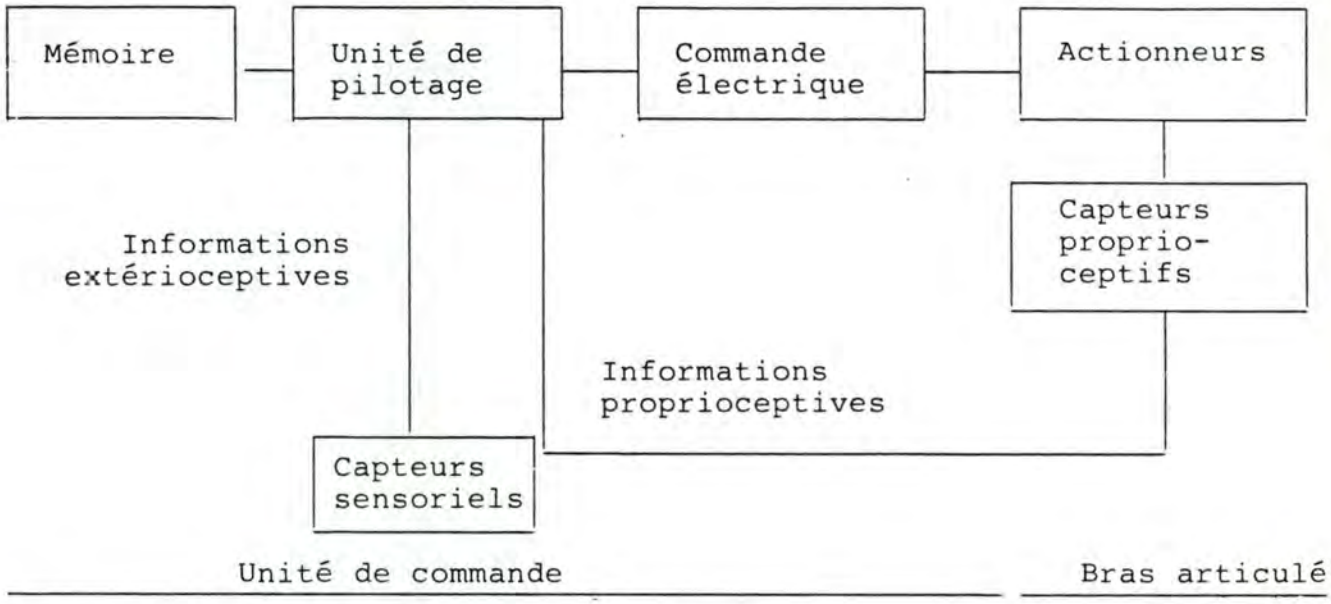



Fig. 2.1 - Composition du robot industriel.

Passons en revue ces différents éléments :

- La mémoire : la mémoire est l'élément du robot qui permet de stocker tous les éléments nécessaires à l'accomplissement d'une tâche. Cette mémoire est dans la majorité des cas une mémoire numérique similaire à celle des ordinateurs.

- L'unité de pilotage : cette unité a pour but de provoquer la commande électrique des actionneurs du bras articulé et d'exécuter ainsi une tâche conforme à sa description stockée dans la mémoire.

- La commande électrique : les sigaux émis par l'unité de pilotage sont des signaux digitaux de faible voltage. Les actionneurs du bras articulé requièrent une source d'énergie puissante (courant électrique, pression hydraulique, pression pneumatique...) pour mobiliser les articulations dans la

...

position dictée par l'unité de pilotage. La commande électrique servira donc d'interface entre la partie informatique et la partie mécanique du robot.

- L'actionneur : un actionneur n'est autre que le moteur d'une des articulations du robot.

- Les capteurs proprioceptifs : ces capteurs sont solidaires des articulations. Ils délivrent une information utilisée exclusivement par l'unité de pilotage que l'on appelle information proprioceptive.

- Informations proprioceptives : informations délivrées par des capteurs solidaires des articulations. Ces informations permettent de contrôler les mouvements du bras articulé car elles donnent à tout instant la position de chacun des corps.

- Capteurs sensoriels : ces capteurs recueillent des informations sur des événements survenus dans l'espace de travail du robot. Il s'agit par exemple de la pression exercée par la pince, de la force exercée par le poignet ou de la détection d'une collision du bras articulé avec un obstacle. Ces informations sont appelées informations extérioceptives.

## 2.2. LES DIFFERENTS TYPES DE PROGRAMMATION

### 2.2.1. La programmation on-line

Dans ce type de programmation, le système de programmation fait partie intégrante de l'unité de commande du robot. La programmation se fait soit en utilisant un boîtier de commande (=teaching box), soit en utilisant un pantin caractérisé par les mêmes contraintes géométriques et mécaniques que le robot (=syntaxeur), soit en actionnant manuellement le bras du robot et en lui faisant accomplir les mouvements qu'il devra reproduire fidèlement par la suite (=teaching by doing). La majorité des robots actuellement installés sont programmés selon l'une de ces trois méthodes.

### 2.2.2. La programmation off-line

Dans ce cas-ci, le robot accomplit une tâche en se basant sur une description formulée à l'aide d'un langage de programmation. Ce langage est caractérisé par une syntaxe propre

au robot. Le robot et son unité de commande deviennnent des unités périphériques d'un ordinateur. Cet ordinateur génère une série de commandes destinées à l'unité de commande. Cette dernière les décode et les exécute en agissant entre autres sur la position des articulations.

Cette programmation est supérieure à la précédente à deux points de vue :

- L'exploitation optimale : un robot coûte cher; il faut donc l'utiliser au maximum de sa capacité. Avec une programmation off-line, on peut programmer de nouvelles applications robotiques pendant que le robot exécute une autre application; pour la programmation on-line, le robot est non productif pendant tout le temps d'apprentissage des mouvements.

- Le comportement adaptatif : la programmation off-line permet d'étendre le champ d'application de la robotique. Un robot muni de capteurs est en mesure de détecter tout événement survenant dans son environnement de travail. Il peut alors faire part de ces événements à l'ordinateur qui le pilote. L'ordinateur pourra alors élaborer une nouvelle stratégie de mouvements tenant compte de ces événements. La programmation off-line permet donc d'exploiter une certaine forme d'"intelligence" dont l'ordinateur est capable. Dans une chaîne de montage, par exemple, certains capteurs pourraient détecter la mauvaise qualité d'une pièce et par conséquent, provoquer la génération des mouvements destinés à la rejeter.

### 2.2.3. Niveaux de la formulation de la tâche d'un robot

(PARENT & LAURGEAU 1983, J. SIMONS 1983)

#### 2.2.3.1. Le niveau articulaire

La formulation d'une tâche au niveau articulation consiste à définir l'ensemble des mouvements que devront effectuer les organes du robot.

Dans le cas qui nous préoccupe, tous les organes sont animés par des mouvements de rotation autour d'un axe commun avec l'extrémité de l'organe précédent de la chaîne. Ces rotations sont assurées par des pivots. D'autres

robots assurent le mouvement de leurs organes par des translations à l'aide d'un mécanisme à coulisse. D'autres robots encore assurent le mouvement d'une partie de leurs organes par translation et de l'autre par rotation.

Tout organe est mobilisé par un actionneur indépendant. La position de chaque organe est donc exprimée à l'aide d'une variable articulaire indépendante qui représente sa rotation ou sa translation par rapport à une position de référence prédeterminée. La valeur de chaque variable articulaire représente un nombre de pas. Ce pas correspond au plus petit angle de rotation ou la plus petite distance de translation que peut subir un organe en agissant sur la coulisse ou sur le pivot auquel il est rattaché. Pour un même robot, le pas n'est pas nécessairement identique pour toutes ses articulations. Enfin, la précision du robot est fonction du choix du pas par le constructeur.

On représente la configuration des organes du robot à l'aide d'un vecteur de N variables articulaires, N étant le nombre d'articulations du bras du robot. Il est donc évident qu'un vecteur de variables articulaires est une représentation de la configuration propre à chaque robot.

La programmation d'un robot au niveau articulaire consistera, en définitive, à préciser la succession des configurations de ses membres en vue d'accomplir une tâche donnée. Cette programmation est donc non portable.

Il se pose encore un autre problème. Nous savons que l'ensemble des organes constitue une chaîne. Par conséquent, tout mouvement d'un organe de la chaîne provoque un changement de position et d'orientation de tous les organes en amont. Il faut donc compenser le déplacement de ces organes en faisant subir un mouvement au premier organe en amont qui à son tour provoque un déplacement des organes qui lui succèdent dans la chaîne.

La programmation du niveau articulaire est donc extrêmement fastidieuse car il faut trouver un compromis entre la position de tous les organes par rapport à leurs

prédécesseurs dans la chaîne pour assurer une maîtrise du déplacement de la pince. On peut donner un cas typique de définition d'un mouvement des organes d'un robot :

avec une vitesse maximale et en continu
amener le bras à la configuration 0, -3075,900,900,30
-800,455,-841,510,0
-800,437,-850,486,0
fermer la pince ...

Le passage d'une configuration à l'autre détermine donc un mouvement. Les variables articulaires représentent des pas exprimés en dixièmes de degrés.

L'interpréteur cablé dans l'unité de commande du robot RM 501 offre également une programmation du niveau articulaire. Le mouvement identique au précédent sera obtenu par

```
SP 9                                (vitesse maximale)
PS 1,0,-4100,3600,1450,-950         (configuration 1)
PS 2,-3204,1823,-3364,-686,686      (configuration 2)
PS 3,-3204,1748,-3401,-649,649      (configuration 3)
MO 0                                (aller à la position 0)
MC 3                                (parcourir successivement
                                     les 3 configurations
                                     qui suivent la confi-
                                     guration 0)
GC                                  (fermer la pince)
```

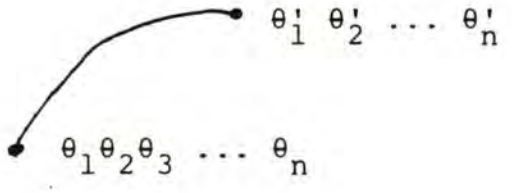
Remarque : les variables articulaires expriment des pas de 0,025 et 0,075 degrés (voir Annexe A)

2.2.3.2. Le niveau XYZ

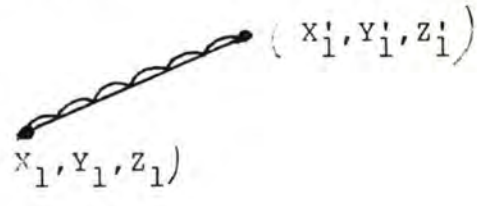
La formulation d'une tâche au niveau XYZ est déjà plus simple que précédemment. Comme son nom l'indique, le problème de la programmation du niveau XYZ se pose dans un espace de travail cartésien à trois dimensions. Le repère de cet espace est souvent associé à la base du robot. On effectue un mouvement en précisant les coordonnées cartésiennes d'un point de l'espace que l'on désire atteindre. Le mouvement sera accompli lorsque les coordonnées cartésiennes de l'extrémité de l'organe terminal du bras articulé coïncideront avec celles du point à atteindre. On remarque donc que seule la position de cet organe terminal nous importe et que les positions des autres organes sont devenues transparentes. Au niveau articulaire, nous avions défini le concept de configuration; au niveau XYZ, il existe un concept similaire. A toute configuration correspond une seule position et orientation de l'extrémité de l'organe terminal. Au niveau XYZ, on spécifie simplement la position et l'orientation de l'organe terminal par rapport au repère de l'espace cartésien. Au niveau XYZ, on peut modifier l'orientation de l'organe terminal indépendamment de sa position; au niveau articulaire, la position et l'orientation de l'organe terminal sont liées à la position des organes en aval de la chaîne.

Les trajectoires décrites par l'organe terminal du robot dans l'espace de travail cartésien sont des trajectoires obéissant à une loi mathématique. Dans notre cas, elles seront rectilignes ou quasiment rectilignes avec une certaine tolérance.

Trajectoire générée lors de l'exécution d'une commande de niveau articulaire




Trajectoire linéaire générée lors de l'exécution d'une commande de niveau "XYZ"




...

On peut donner un exemple typique de programme du niveau XYZ :

avec une tolérance de 2mm et en continu

amener la pince en $(X_1\ Y_1\ Z_1\ ,\ _1\ _1\ _1)$

amener la pince en $(X_2\ Y_2\ Z_2\ ,\ _2\ _2\ _2)$

fermer la pince

$X_i$, $Y_i$, $Z_i$ représentent l'abscisse, l'ordonnée et la hauteur de l'extrémité de la pince.

$_i\ ,\ _i\ ,\ _i$ représentent l'orientation de la pince. Nous préciserons ultérieurement la signification de ces paramètres.

2.2.3.3. Le niveau "objet"

Au niveau "objet", on utilise une base de données des objets se situant dans l'espace de travail du robot. Cette base de données donne les relations géométriques qui lient les objets entre eux précisant leurs positions, leurs orientations et leurs caractériques géométriques, la manière de les approcher, de les serrer ...

Cette base de données donne également la description de certaines positions fixes de l'espace de travail et précise la manière de les approcher. J'appelle ces positions fixes des sites.

Les informations contenues dans cette base de données permettront d'améliorer,entre autres, la lisibilité et la souplesse de la programmation.

Au niveau XYZ, on définissait les mouvements en termes de positions à atteindre. Il est donc difficile de comprendre l'objectif d'une tâche si l'on ne peut faire la correspondance de ces positions avec les objets de l'espace de travail. La formulation d'une tâche à ce niveau consiste à préciser uniquement les mouvements que les objets doivent subir. Par exemple :
- déplacer l'objet A sur l'objet B;
- saisir l'objet C et le placer au site X de l'espace de travail.

...

Le fait de nommer les objets et les sites au lieu de donner leur position améliore la lisibilité de la programmation.

Les programmes du niveau "objet" sont aussi plus souples car toute modification de la position initiale des objets et des sites ne nécessite pas de modifications de la programmation.

Le niveau "objet" assure par ailleurs la mise à jour automatique de la base de données après le déplacement de tout objet.

On spécifie la tâche à accomplir mais pas la manière dont il faut l'accomplir. Le problème à résoudre à ce niveau consistera à générer la trajectoire automatiquement. Au niveau "objet", on ne tient pas nécessairement compte de la présence d'éventuels obstacles sur la trajectoire calculée. On risque alors de provoquer des collisions entraînant un déplacement involontaire de certains objets de l'espace de travail. Le modèle de l'espace de travail dans la base de données pourrait ne plus coïncider avec sa configuration réelle. Pour éviter ce problème, il y a parfois moyen de disposer les objets de manière à éviter ce genre de collision. Cette solution n'est pas toujours possible surtout lorsque l'espace de travail est particulièrement encombré. De plus, cette solution n'est pas très élégante parce qu'elle nécessite de la part du programmeur une connaissance bien précise de la manière dont les trajectoires sont générées (souvent ces trajectoires sont rectilignes mais ce n'est pas une règle impérative).

Certains langages du niveau "objet" permettent de préciser des points intermédiaires par lesquels la trajectoire générée devra obligatoirement passer. On aura des instructions de la forme :

```
déplacer l'objet A
        via le site B
        via le site C
              |
        sur l'objet X
```

Cette solution permet de résoudre le problème de collisions mais la génération de la trajectoire n'est plus tout à fait automatique.

...

2.2.3.4. Le niveau "tâche"

Ce niveau répond au problème posé par le niveau précédent. Le programmeur ne devra donc plus détailler les étapes intermédiaires de la trajectoire en vue d'accomplir un objectif donné. La trajectoire de déplacement des objets est calculée automatiquement par un planificateur de trajectoires. Ce planificateur se sert d'un modèle de l'espace de travail. Ce modèle lui permet de détecter les obstacles qui se trouveraient sur la trajectoire des objets à déplacer et de générer des trajectoires de contournement.

Bien qu'il existe une littérature abondante sur la planification des trajectoires, presque aucun robot ne fonctionne selon ce mode d'exécution. La programmation au niveau "tâche" correspond en quelque sorte à une programmation automatique à partir d'un langage de très haut niveau; c'est donc la plus simple et la plus puissante des programmations.

# 3. DEFINITION DU PROJET

## 3.1. Objectif

Le coût de programmation d'un robot représente une grande partie du coût de fonctionnement globale d'un atelier flexible".

Dans certains cas, le coût de la programmation peut dépasser de loin le coût d'achat du matériel. Le même problème s'était déjà posé pour la programmation des ordinateurs. La programmation en assembleur, bien qu'efficace, coûte extrêmement cher, car le coût de développement et de maintenance sont considérables. Par ailleurs, ces programmes sont spécifiques à une machine donnée et ne sont donc pas transportables sur d'autres machines. Pour diminuer le coût de la programmation, on a inventé des langages de haut niveau. Ces langages de haut niveau ont un double intérêt :

- -ils permettent de simplifier fortement la tâche de la programmation;
- -les programmes écrits en langage de haut niveau sont indépendants des caractéristiques de la machine sur laquelle ils sont développés.

Par analogie avec la programmation classique, la programmation d'un robot au niveau articulaire correspond à la programmation en assembleur. Elle est de loin la plus efficace mais n'est en aucun cas transportable car elle est fonction des caractéristiques géométriques (nombre de degrés de liberté, longueur des éléments du bras) et mécaniques (amplitude des mouvements de chaque articulation) du robot. De plus, la syntaxe du langage de commande offert par le constructeur n'est pas identique pour tous les robots.

Une programmation de haut niveau s'avère donc également nécessaire en robotique. Une programmation simple pour la réalisation d'une tâche complexe, une meilleure portabilité, une meilleure modifiabilité, une indépendance par rapport aux caractéristiques du robot devraient permettre de diminuer les coûts tout en étendant les champs d'application.

Nous avons vu qu'il y avait deux grandes catégories de langage de programmation : - la catégorie "robot-level"
- la catégorie "task-level"

...

Nous savons que la catégorie "robot-level" est utilisée pour la programmation de la majorité des robots de la première et de la deuxième génération. Nous savons aussi que la programmation des robots à l'aide des langages de cette catégorie requiert de la part du programmeur une certaine expérience en informatique et dans l'utilisation des capteurs sensoriels éventuels. Cette difficulté à programmer provient du fait que les caractéristiques du robot ne sont pas transparentes aux yeux du programmeur, car la description de la tâche exige une descritpion explicite de tous les mouvements du robot en vue de l'accomplir.

Nous allons nous tourner vers la deuxième catégorie de langage de programmation. D'abord parce qu'elle permet par sa simplicité à un non-initié de programmer un robot. Ensuite, parce qu'elle permet une meilleure portabilité de la programmation sur d'autres installations car, les caractéristiques physiques du robot y sont transparentes.

Nous avons dit qu'il y a presque autant de langages de programmation robotique qu'il y a de constructeurs. Ces langages sont pourtant proches des langages traditionnels de programmation par leur possibilité de structurer les données et les traitements.

Il serait peut être nécessaire de bâtir un environnement de programmation qui permette de programmer différents robots à l'aide d'un même langage. Cet environnement de programmation permettra une programmation de niveau "tâche". Il permettra l'utilisation d'un langage classique de programmation pour définir une tâche à accomplir, indépendemment du robot utilisé. L'usage d'un même langage de programmation pour tous les robots permettra d'assurer une meilleure portabilité de la programmation des applications robotiques.

La traduction des applications en une suite de commandes caractérisée par la même syntaxe que celle du langage défini par le constructeur devra être effectuée automatiquement par l'environnement de programmation.

Enfin, nous avons vu que la grande majorité des robots actuellement en fonctionnement étaient des robots de la première génération.

L'environnement de programmation sera donc spécialement conçu pour la programmation des robots de la première génération. Nous nous préoccuperons particulièrement de la programmation d'un petit robot industriel commercialisé par le constructeur Mitsubishi. Le langage de programmation défini par le constructeur est un langage d'un niveau articulaire.

## 3.2. Présentation du robot à programmer

Le robot à programmer est un produit Mitsubishi de la série RM 501 Movemaster. Il répond à la définition du robot industriel énoncée précédemment. Le bras articulé est composé d'une chaîne de cinq organes en série reliés entre eux par cinq articulations indépendantes.

### 3.2.1. Les caractéristiques du langage offert par le constructeur

Nous avons déjà signalé que le constructeur offrait un langage de niveau articulaire pour la programmations du robot Movemaster RM 501.

L'annexe A de ce travail contient la description des commandes utilisés par les environnements de programmation pour assurer le pilotage de ce robot.

### 3.2.2. Les organes du robot

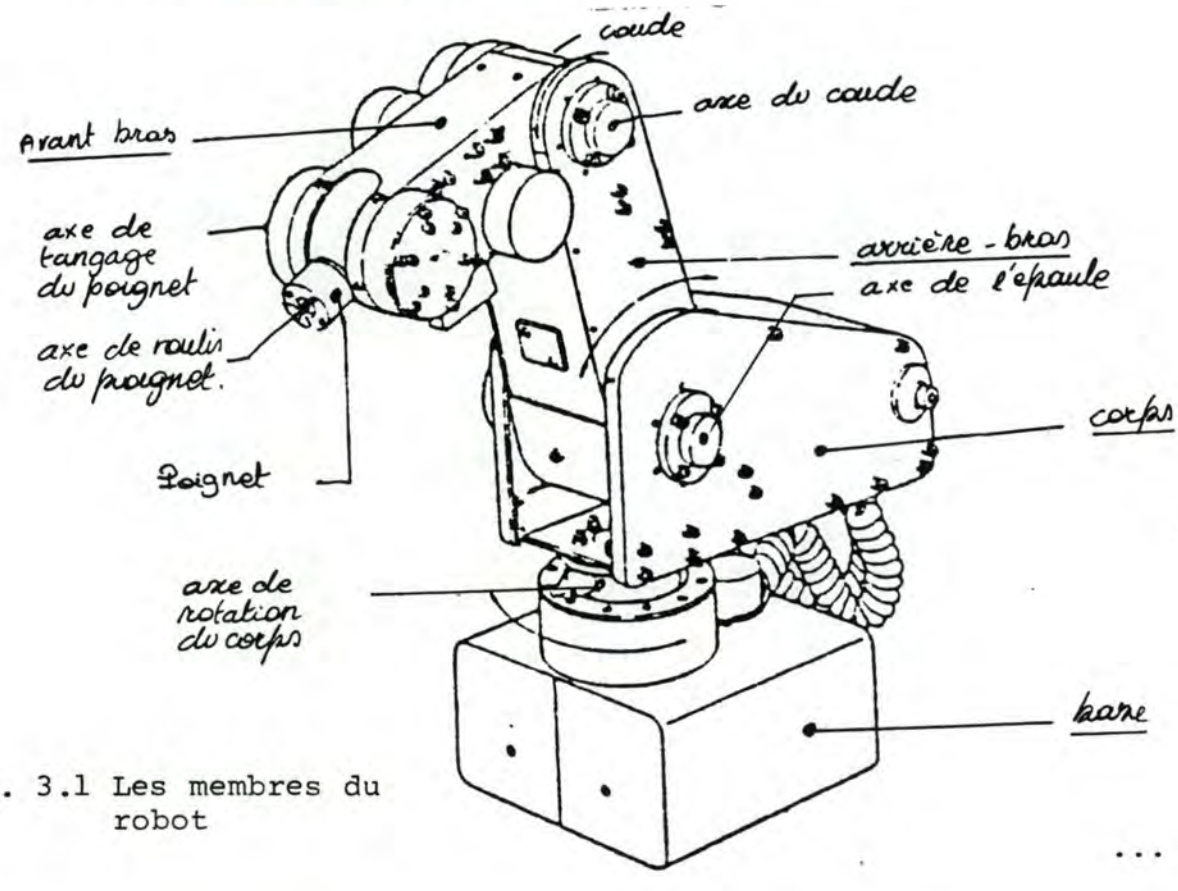



Fig. 3.1 Les membres du robot

...

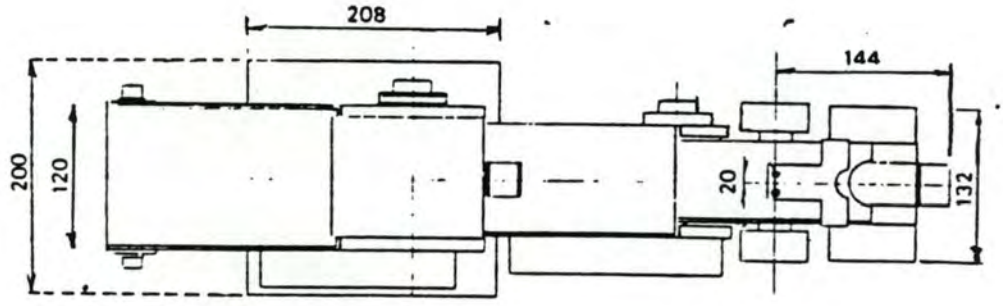
### 3.2.3. Amplitude des mouvements des différents organes du bras articulé

| | | |
|---|---|---|
| Rotation du corps | | 300° |
| Rotation de l'épaule | | 130° |
| Rotation du coude | | 90° |
| Inclinaison du poignet | ± | 90° |
| Torsion du poignet | ± | 180° |

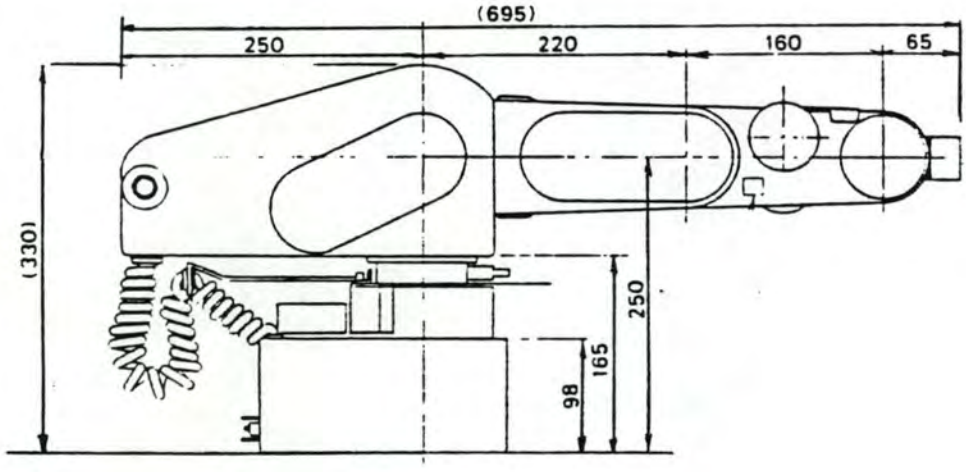
### 3.2.4. Les dimensions des organes

Les deux figures ci-dessous représentent les dimensions des cinq organes du robot. Les mesures sont exprimées en millimètres.

Vue de haut des organes constitutifs du robot




Vue de profil des organes constitutifs du robot




...

Vue de profil de la pince std. Vue de face la pince stand.

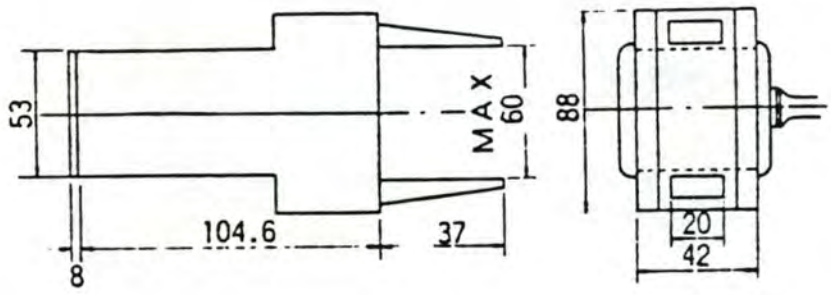



Fig. 3.2 - Dimensions des organes du robot

3.2.5. Dimensions du volume accessible par le robot

Les figures ci-dessous représentent une vue de profil et de haut de l'espace accessible par le poignet du robot. On peut fixer une pince standard sur le site d'attache situé à l'extrémité du poignet. L'espace accessible par cette pince ne sera pas identique à l'espace accessible représenté par les figures, il aura toutefois des dimensions comparables à celles exprimées par la figure.

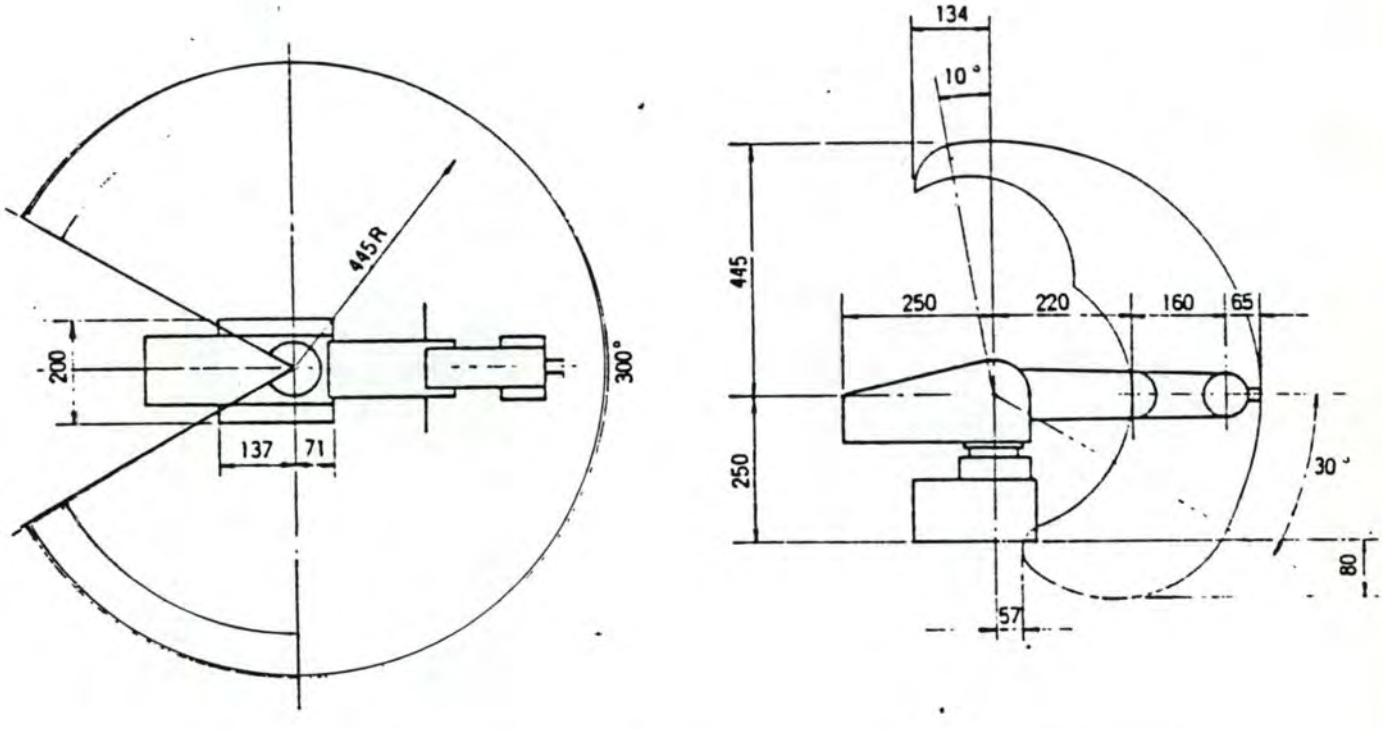



Fig. 3.3 Dimensions du volume accessible par le robot

...

## 3.3. Architecture globale du produit logiciel

La hiérarchie des niveaux de la formulation de la tâche d'un robot énoncée au paragraphe 2.2.3., nous donne déjà une idée assez nette de la structure globale du produit logiciel à développer. On dispose d'un robot dont la programmation est du niveau le plus élémentaire (niveau articulaire) et on désire formuler la tâche qu'il devra exécuter à l'aide d'un langage de haut niveau (niveau tâche). On peut transformer successivement la formulation de la tâche en une formulation de niveau inférieur pour enfin envoyer au robot, des commandes du niveau articulaire. Chaque étape de cette transformation sera effectuée par une couche. A chaque couche correspond un niveau de programmation.

Passons ces couches en revue.

### La couche articulaire

Cette couche sert,d'une part, à gérer des trajectoires des organes articulés du bras et à assurer une indépendance des programmes du niveau articulaire par rapport aux particularités syntaxiques du langage de niveau articulaire défini par le constructeur.

La description d'une tâche au niveau articulaire consiste à définir une succession des configurations des organes du robot. Le passage successif d'une configuration à l'autre avec une vitesse douce définira un mouvement.

La couche articulaire fait apparaître le concept de robot articulaire. Ce robot est caractérisé par la même géométrie et la même mécanique que le robot à programmer mais la syntaxe de sa programmation est indépendante de celle du robot à programmer.

### La couche "XYZ"

La couche "XYZ" comporte un transformateur de coordonnées qui a pour but de calculer la configuration des organes du robot articulaire qui permet de positionner la pince à une position cartésienne donnée et avec une orientation donnée.

Cette couche permet d'ignorer les caractéristiques géométriques du robot à programmer, car elle nous permet de nous

préoccuper seulement de la position de l'extrémité de la pince et plus de la position et de l'orientation des organes qui la portent. Un programme du niveau "XYZ" sera vu par cette couche comme le passage successif d'une position cartésienne à l'autre de la pince dans l'espace de travail.

La couche fait apparaître le concept de robot XYZ. Ce robot ne dépend plus des caractéristiques géométriques du robot à programmer.

La couche de niveau "objet"

Cette couche contient une description de tous les objets disposés dans l'espace de travail; elle assure la mise à jour et la création de cette description.

La programmation à ce niveau est vue comme une succession de déplacements des objets de l'espace de travail. Il faut toutefois préciser une partie de la trajectoire de déplacement de ces objets.

La couche de niveau "tâche"

La couche de niveau "tâche" contient un planificateur de trajectoires qui sera chargé de déterminer la trajectoire de déplacement des objets de l'espace de travail, de manière à ce que ce déplacement ne provoque pas de collision avec les autres objets de l'espace de travail. Ce niveau permet une programmation au niveau "tâche".

Relation entre les couches

La structure globale du produit logiciel sera une structure classique basée sur une relation d'utilisation.

- La couche de niveau tâche utilise un modèle géométrique de l'espace de travail pour déterminer les trajectoires sans collision. Il fera donc appel aux services offerts par la couche du niveau objet qui est chargé de créer et de mettre à jour le modèle dont le planificateur de trajectoire a besoin.
- La couche de niveau objet utilise les services offerts par la couche "XYZ", cette dernière permet de déplacer les objets de l'espace de travail selon des trajectoires rectilignes. La définition de cette trajectoire n'est

possible que si la pince peut passer successivement par des positions cartésiennes calculées. Elle fera donc appel aux services offerts par la couche "XYZ".

- La couche de niveau "XYZ" est chargée de calculer l'angle de rotation de chacune des articulations du robot. Connaissant la position et l'orientation de la pince dans l'espace de travail, le niveau "XYZ" connaît les caractéristiques géométriques du robot à programmer mais ne connaît par contre, pas les caractéristiques de son langage de commande. De plus, il n'est pas en mesure de détecter les inversions brusques du sens de rotation des articulations lors du passage en continu par plusieurs configurations successives. La couche articulaire se chargera de la traduction des commandes émises par la couche "XYZ" en commandes spécifiques au robot à programmer. Elle se chargera par ailleurs de la gestion des trajectoires car cette fonction n'est pas assumée par la couche "XYZ"

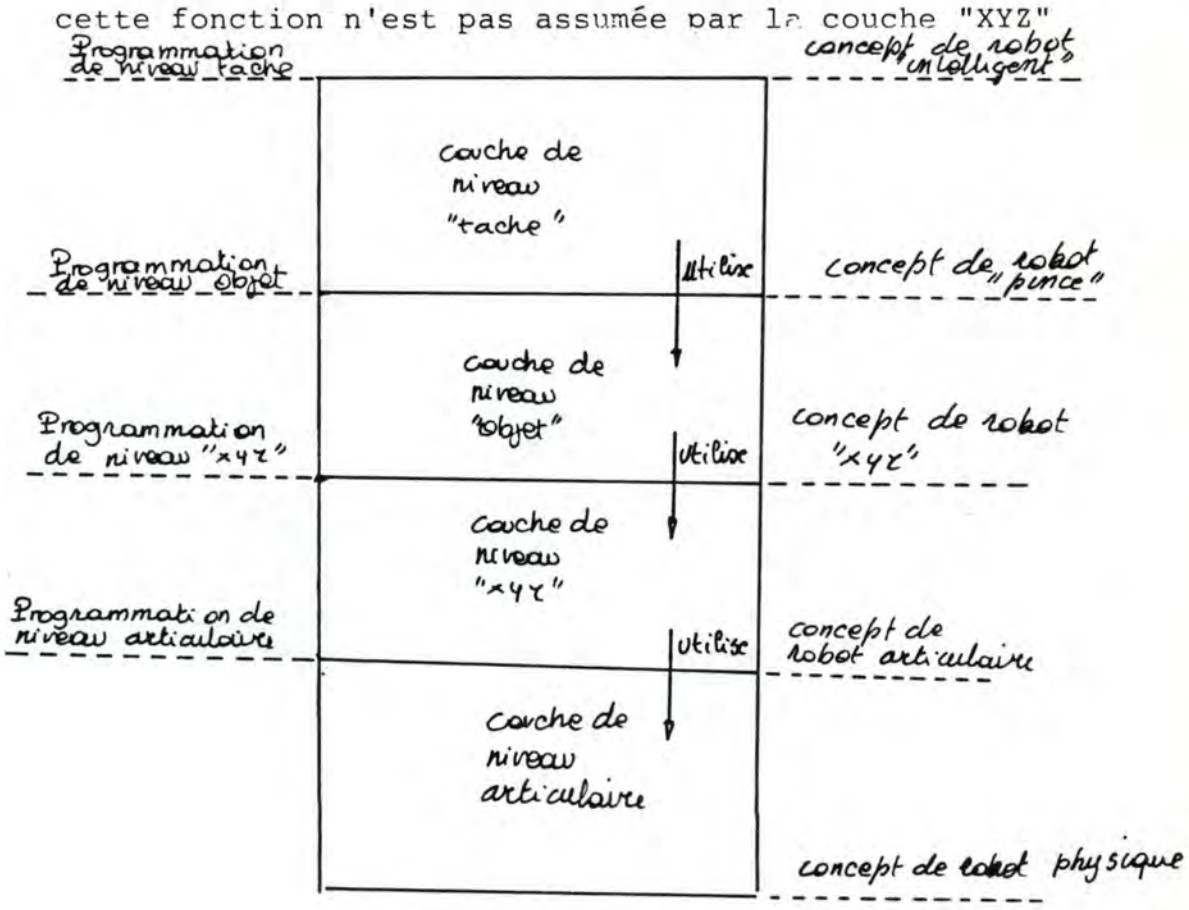



Fig.3.4 - Architecture globale du produit logiciel

...

## 3.4. L'utilisation de Prolog

- Choix de Prolog comme langage de programmation du robot

Les personnes chargées de la programmation d'un robot ne seront pas nécessairement des experts en programmation. Il faut donc mettre à leur disposition un langage de programmation qui est à la fois clair et facile à utiliser.

La clarté : déjà, la plupart des langages de programmation classiques sont caractérisés par une syntaxe quelque peu ésotérique. Cette fâcheuse tendance est encore plus nette pour les langages de programmation spécifiques à la robotique. L'annexe A contient la description d'un de ces langages de programmation. Nous avons retenu Prolog comme langage de programmation pour sa clarté et la simplicité de sa syntaxe.

Facilité d'utilisation : il existe deux tendances en programmation classique.

L'usage d'un langage compilé : les langages compilés sont de loin plus efficaces mais il faut reconnaître que leur utilisation n'est pas toujours facile. Pour exécuter un programme, il faut compiler sa version texte, soumettre le résultat à "l'éditeur de liens", charger en mémoire la version exécutable du programme et enfin, lancer l'exécution. Il se pose également un problème en cas d'erreur d'exécution. A cause de la traduction du contenu du fichier texte, il est difficile d'y localiser les erreurs qui ont été détectées lors de l'exécution de sa version "exécutable".

L'usage d'un langage interprété : l'usage d'un langage interprété est plus simple. Les expressions du langage seront directement exécutées par la machine. On résout de cette manière le problème de la localisation des erreurs de programmation et on simplifie par la même occasion le mode d'exécution car la traduction de la source n'est plus nécessaire.

Enfin, il faut dire que c'est quand même par la force des choses que nous avons choisi Prolog comme langage de programmation des applications robotiques car l'environnement de programmation est lui-même écrit en Prolog. Il met à la disposition de l'utilisateur un ensemble de primitives

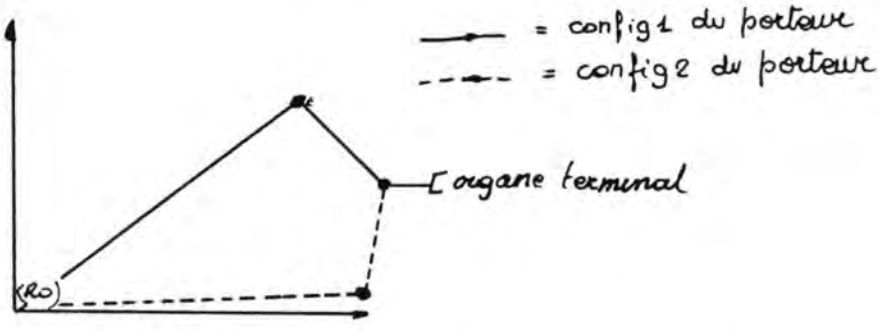
de programmation qui sont en fait des prédicats Prolog. Ils ne pourront donc être utilisés que dans un programme Prolog.

- Choix de Prolog comme langage d'implémentation de l'environnement de programmation

Nous avons vu que le logiciel était structuré en quatre couches. Chaque couche est implémentée en Prolog mais dans chacune de ces couches, l'usage de Prolog y est différent.

La couche articulaire : les problèmes traités dans la couche articulaire sont purement déterministes. L'usage de Prolog ne s'y justifie donc pas car on y sous-exploite ses capacités. Il était toutefois intéressant d'utiliser ce langage pour l'implémentation de cette couche, ne fut-ce que pour prouver qu'il convient également pour traiter des problèmes déterministes.

La couche "XYZ" : elle sera chargée entre-autre, de calculer la position des corps du robot de manière à positionner la pince à une position cartésienne donnée. Il arrive qu'il y ait plusieurs solutions possibles de positionnement des organes du robot pour une même position de la pince. Il peut même arriver que le nombre de solutions soit infini. Le manipulateur ci-dessous est constitué de trois organes. Pour une même position de l'organe terminal, il y a moyen de positionner les organes du porteur de deux manières différentes.




Le problème n'est pas déterministe car le nombre de solutions admissibles n'est pas connu à l'avance et il faudra de plus, sélectionner une de ces solutions. Ce genre de problème est traité très facilement à l'aide d'un langage tel que Prolog.

...

La couche "objet" : cette couche se charge notamment de la création et de la mise à jour d'une base de données des objets de l'espace de travail du robot. A nouveau, Prolog convient particulièrement bien, car ce langage dispose de mécanismes qui permettent de créer facilement un petit système de gestion de base de données relationnelle en mémoire.

La couche "tâche" : cette couche contient un planificateur de trajectoire basé sur la méthode "generate and test" Le planificateur de trajectoire est une fonction de décision évoluée. Elle est caractérisée par l'aptitude à réagir correctement à des situations concrètes pour lesquelles il n'a pas été explicitement programmé. Le planificateur que nous avons défini au chapitre 8 utilise pleinement les mécanismes d'unification et de backtracking ainsi que la récursivité offerts par le langage Prolog.

## 3.5. Etat d'avancement du projet

Le projet est très ambitieux, il n'a donc pas été possible de le terminer. Toutefois, les couches articulaires et "XYZ" ont été implémentées. Nous avons déjà défini une série de programmes d'applications de niveau "XYZ". Ces programmes figurent au chapitre 6. Les couches objet et tâche ont été seulement spécifiées. Il reste donc à les implémenter.

# 4. LA COUCHE ARTICULAIRE

La couche de l'articulaire remplit trois rôles :

1) Elle assure une indépendance des programmes de niveau articulaire par rapport aux particularités syntaxiques du langage de commande du robot à programmer.

2) Elle assure une gestion des mouvements des organes du bras articulé lors du passage d'une configuration à l'autre de manière
   - à protéger le mécanisme des chocs dûs à des freinages brusques ou des inversions instantanées du sens de rotation des actionneurs:
   - à rendre transparentes les limites de stockage de la mémoire de l'unité de commande du robot.

3) Elle permet de choisir le mode de programmation et d'exécution du programme généré pour le robot. L'unité de commande du robot à programmer contient un interpréteur de programme de niveau articulaire. Cet interpréteur ressemble étrangement à un interpréteur Basic. On y retrouve d'ailleurs ses deux modes d'exécution.

   a) le mode d'exécution immédiat : l'interpréteur exécute les instructions au fur et à mesure de leur introduction dans la mémoire. Pour qu'une instruction soit exécutée immédiatement, il suffit qu'elle ne soit pas précédée d'un numéro de ligne. L'avantage de ce mode d'exécution réside dans le fait que l'on n'est jamais confronté au problème de la limite de stockage de la mémoire.
   Par contre, ce mode d'exécution coûte cher en temps de calcul car pour chaque exécution d'un même programme, la couche de l'actionneur enverra systématiquement la même suite de commandes à l'unité de commande du robot.
   Ce mode d'exécution est toutefois très souple. Si le robot physique est muni de capteurs sensoriels, son comportement peut être adaptatif. En effet, il peut signaler la survenance d'événements à l'ordinateur qui le pilote. Cet ordinateur répondra par une séquence de commandes spécifiques à l'événement.

b) le mode différé : la totalité du programme est alors chargée en mémoire avant son exécution. Toute son instruction doit alors être précédée d'un numéro de ligne. Le programme sera exécuté dès l'envoi de la commande immédiate "RUN". Ce mode d'exécution a exactement les avantages et les désavantages inverses de ceux du mode d'exécution précédent.

c) le mode mixte : ce mode mixte concilie en partie les avantages des deux modes précédents.
   - dans le cas qui nous occupe, le robot à programmer est dépourvu de capteurs sensoriels. Par conséquent, N exécutions d'une même application provoquera rigoureusement les mêmes mouvements du bras. Il est donc inutile de générer N fois une suite de commandes identiques.
   - ensuite la taille de la mémoire est limitée, ce qui constitue une contrainte pour l'application du deuxième mode d'exécution.

   La solution mixte consistera donc à différer l'envoi des commandes au robot en les envoyant dans un fichier texte. Le contenu de ce fichier sera donc un scénario de travail qui pourra être transféré au robot autant de fois qu'on le désire.

   Dans notre cas, ce dernier mode est de loin le plus efficace. L'environnement de programmation pourrait être considéré comme une sorte de compilateur de programmes robotiques de haut niveau.

La couche articulaire comporte une série d'objets. Chaque objet est caractérisé par une liste de compétences. Nous définirons ces objets puis leurs compétences. Par ailleurs, la couche articulaire met en relation deux concepts : le concept de robot physique et le concept de robot articulaire.

- Le robot physique est le robot industriel que l'on désire programmer. Dans notre cas, il s'agit du RM 501 MOVEMASTER de MITSUBISHI. Le robot physique est aussi un objet mais il appartient à la couche inférieure.

- Le robot articulaire est un robot idéalisé car sa programmation est indépendante des particularités syntaxiques du langage de programmation du robot physique. Il est, par contre, caractérisé par la même géométrie et les mêmes contraintes mécaniques que celles du robot physique.

Ces deux robots communiquent par objets interposés. Le robot articulaire envoit des messages aux objets définis dans la couche articulaire. Ces messages activent une compétence des objets cibles. L'activation des compétences de certains objets provoque l'envoi d'un message au robot physique.

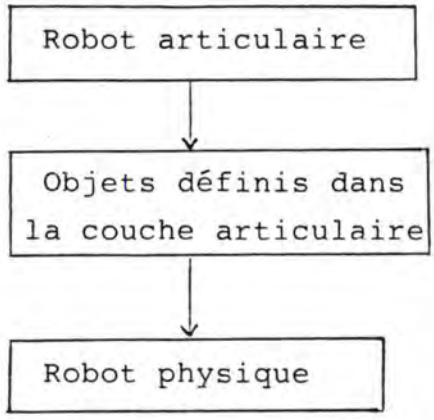



Fig. n° 4.1 - communication du robot articulaire avec le robot physique

## 4.1. Objets définis dans la couche articulaire

La couche articulaire comporte un ensemble d'objets caractérisés par des compétences. Ces objets sont représentés dans la figure 4.2. Chaque objet y est encadré. Les arcs qui relient des objets expriment une relation entre ces objets. La sémantique de toute relation est écrite explicitement sur tout arc.

La liste des compétences qui caractérise chaque objet est écrite sur le côté gauche de chaque cadre qui entoure son nom.

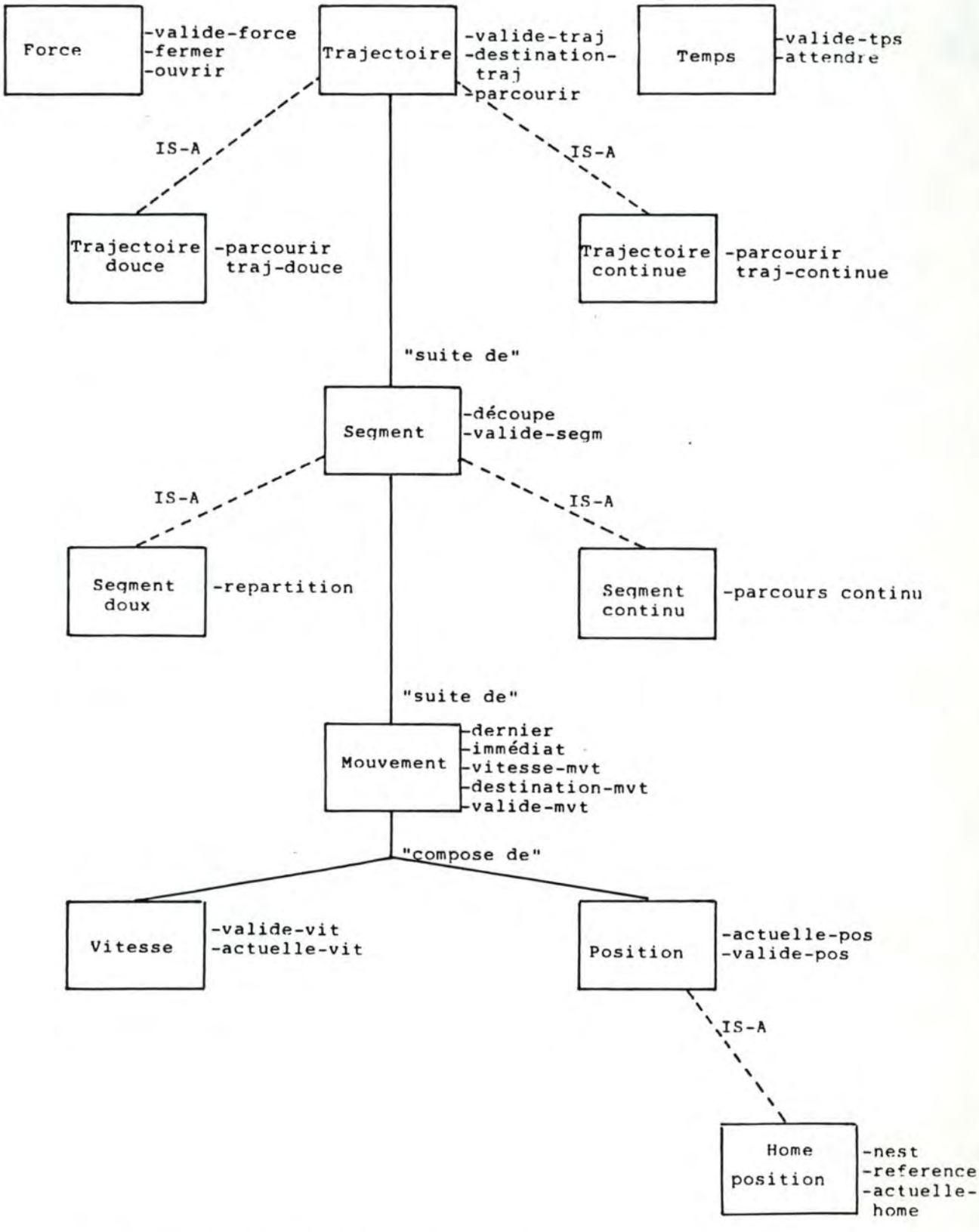



Fig. 4.2 - Schéma des objets définis dans la couche articulaire.

...

## 4.2. Définition des objets

- La position : le concept de position est un concept central de la programmation au niveau articulaire. Un robot n'a de sens que si on peut le positionner de plusieurs manières consécutives en vue d'accomplir une tâche dans le volume de son espace de travail. Une position donnée se traduit par une et une seule configuration des organes du SMA (Système Mécanique Articulé). Cette configuration est représentée par un vecteur de n variables articulaires, n étant le nombre d'articulations indépendantes du SMA. Chaque varialbe articulaires est associée à une articulation et exprime son angle de rotation. La position n'a de sens que si elle est exprimée par rapport à une position de référence appelée "Home position".
- La "Home position" : est une position particulière de référence par rapport à laquelle on exprime les positions du SMA. Ainsi, le robot est positionné à la "home position" lorsque la valeur des variables articulaires exprimant sa position sont toutes nulles.
  - La home position par défaut correspond à l'origine mécanique du SMA.
  - Une nouvelle home position est une position. Elle s'exprime donc par rapport à la home position qu'elle va remplacer.
- La vitesse : exprime une vitesse maximale de translation de l'extrémité du bras lorsque celui-ci passe d'une position à l'autre. Cette vitesse varie entre 40 et 400 mm/s.
- Le mouvement : est un déplacement du robot articulaire vers une position cible avec vitesse déterminée. Le mouvement est donc défini à l'aide de concepts de vitesse et de position définis précédemment.
- Le segment : suite non vide de mouvements.
- Le segment doux : segment dont l'exécution "en continu" des mouvements qui le composent ne provoque pas de choc dû à l'inversion brusque du sens de rotation de l'un des actionneurs.

- Le segment continu : même concept que le segment doux à ceci près que les mouvements du segment doivent être en nombre suffisamment restreint pour ne pas excéder la capacité de stockage de la mémoire de l'unité de commande du robot.
- La trajectoire : suite non vide de segments.
- La trajectoire douce : suite non vide de segments doux.
- La trajectoire continue : suite non vide de segments continus.
- La force : le concept de force désigne la pression qu'exercent les mâchoires de la pince : lors de sa fermeture, pour saisir un objet de l'espace de travail, ou lors de son ouverture, pour produire un écartement des pinces. Le concept de force est défini à l'aide de trois paramètres. Les paramètres $a_1$, $a_2$ indiquent un ampérage qui se traduit par une pression et la troisième $a_3$ indique le temps t pendant lequel la première pression est exercée.

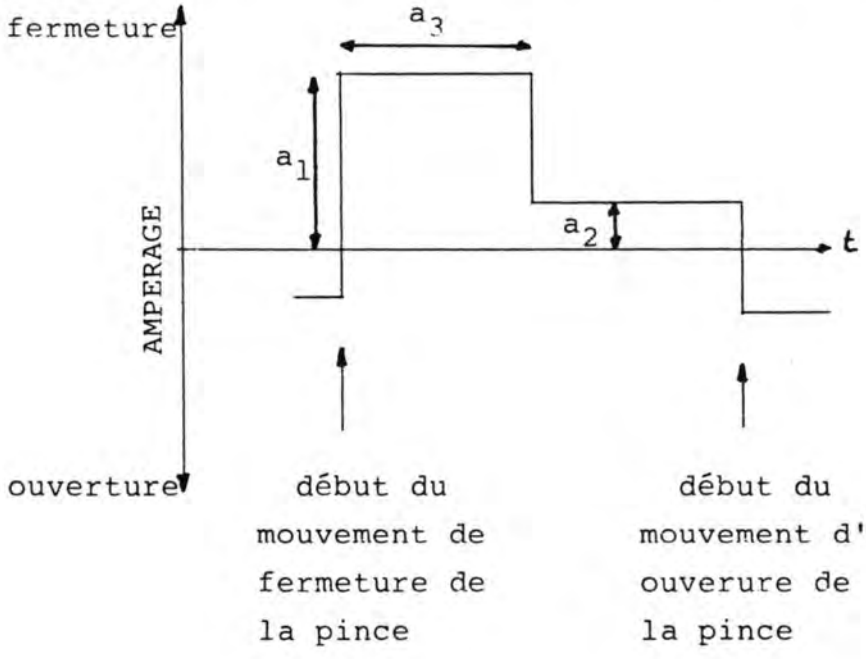



Fig. 4.3 - Représentation de la force exercée par la pince pendant et après sa fermeture jusqu'à son ouverture.

...

Le paramètre $a_1$ représente la pression qui est exercée par la pince pendant une durée exprimée en dixièmes de secondes par le paramètre $a_3$. Cette durée n'est autre que le temps écoulé depuis le début du mouvement de fermeture (ou d'ouverture) de la pince. Le paramètre $a_2$ représente la pression qui est exercée par la pince après l'écoulement de la durée exprimée par le paramètre $a_3$. La force représentée par le paramètre $a_2$ est exercée jusqu'à ce que les mâchoires de la pince subissent un mouvement d'ouverture (ou de fermeture).

- Le temps : est un objet qui représente une durée exprimée en dixièmes de secondes.

## 4.3. Compétences des objets

Dans la figure 4.2, les compétences de chaque objet sont écrites à côté du cadre entourant le nom de chaque classe d'objets. Passons en revue les compétences de chaque classe.

### 4.3.1. Compétences de la position

Actuelle-pos
Objet créé : position
Effet : crée une instance de position représentant la configuration actuelle des corps du robot articulaire.

Valide-pos
Effet : teste la validité de la position. Une position est valide si elle correspond à une configuration des corps du robot compatible avec son mécanisme.

### 4.3.2. Compétences de la home position

Nest
Effet : envoie un message au robot physique qui a pour effet d'initialiser ou de réinitialiser le hardware du robot. Dans les deux cas, les membres du robot sont positionnés à l'origine mécanique. Dans le cas de l'initialisation, la home position correspond toujours à l'origine mécanique.

...

Référence

Effet : modifie la home position du robot articulaire et envoit un message au robot physique qui a pour effet de modifier également sa home position.

Actuelle-Home

Objet créé : position

Effet : retourne la position courante de la "home position" exprimée par rapport à l'origine mécanique du robot.

4.3.3. Compétences de la vitesse

Valide-vit : teste la validité de la vitesse. Une vitesse est valide lorsqu'elle est supérieure ou égale à 40 mm/s et inférieure ou égale à 400 mm/s.

Actuelle-vit : crée et renvoit une instance d'objet vitesse. Cette vitesse représente la vitesse du dernier mouvement effectué par la pince.

4.3.4. Compétences du mouvement

Dernier

Effet : crée et renvoit une instance de mouvement qui représente le dernier mouvement effectué par la pince. En réalité, ce mouvement est construit à l'aide des objets vitesse et position obtenus par l'activation des compétences "actuelle" de la position et "vit-actuelle" de la vitesse.

Immédiat

Effet : provoque l'envoi d'un message au robot physique afin qu'il exécute le mouvement décrit par l'instance de mouvement cible du message.

Vitesse-mvt

Objet créé : vitesse

Effet : crée et renvoit un objet vitesse représentant la vitesse de l'instance de mouvement cible du message.

...

Destination-mvt

Objet créé : position

Effet : crée et renvoit un objet position qui représente la position qu'aurait le robot articulaire.

Valide-mvt

Effet : teste la validité de l'instance de mouvement cible du message. Un mouvement est valide lorsque les objets vitesse et position qui le composent sont valides. On active donc les compétences respectives "valide-vit" et "valide-pos" de ces deux objets pour tester la validité du mouvement.

4.3.5. Compétence du segment continu

Parcours_continu

Effet : l'activation de cette compétence provoque la génération d'une suite de messages destinés au robot physique. Ces messages ont pour effet de mémoriser la suite des mouvements du segment continu dans la mémoire du robot physique. Ces mouvements seront effectués "en continu" par les organes du robot physique.

4.3.6. Compétence de la trajectoire continue

Parcourir-traj-continue

Effet : active la compétence "parcours continu" pour tous les segments continus successifs qui constituent l'instance cible du message.

4.3.7. Compétence du segment doux

Répartition

Effet : subdivise le segment doux cible du message en une suite de segments si ce segment doux est "trop long" pour être un segment continu. La subdivision s'opère si possible aux endroits où les mouvements sont caractérisés par la vitesse la plus faible. (voir fig. 4.5 page 4.10)

...

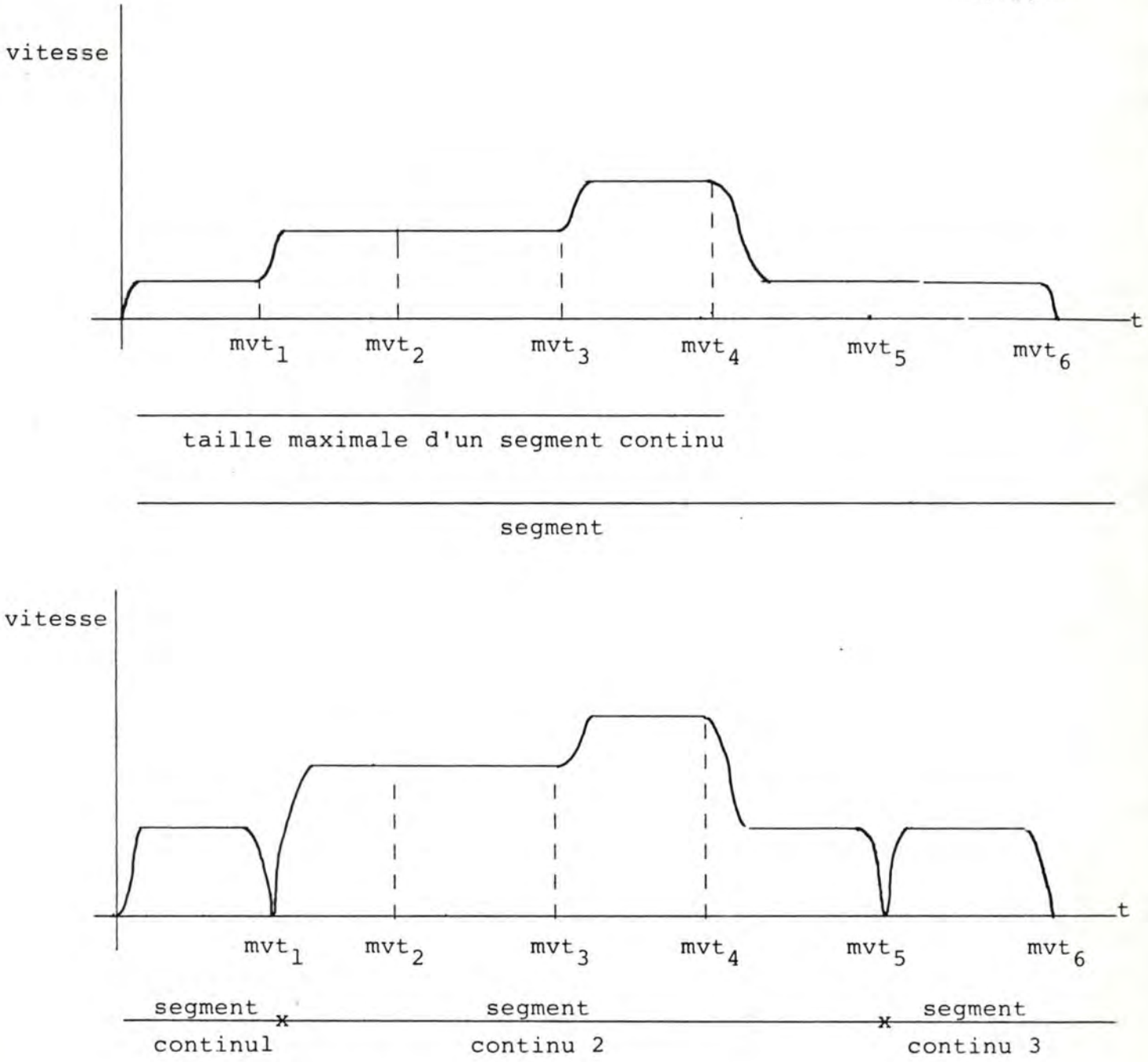



Fig. 4.5 - exemple de répartition.

4.3.8. Compétence du segment

Découpe

Objet créé : trajectoire douce

Effet : subdivise le segment cible du message en une suite de segments doux. La trajectoire douce renvoyée est constituée de cette suite de segments doux.

Valide-segm

Effet : teste la validité du segment. Un segment est valide lorsque tous les mouvements

...

qui le constituent sont valides. Active donc la compétence "valid-mvt" de tous les mouvements du segment.

4.3.9. Compétence de la trajectoire douce

Parcourir-traj-douce

Effet : active la compétence "répartition" des segments doux qui constituent la trajectoire. douce, cible du message. Active la compétence "parcourir-traj-continue" des trajectoires continues obtenues par les activations successives de la compétence "répartition".

4.3.10. Compétences de la trajectoire

Parcourir-traj

Effet : active la compétence "découpe" du seul segment qui constitue la trajectoire cible du message. Active la compétence "parcourir-traj-douce" de la trajectoire douce obtenue par l'activation de la compétence "découpe".

Destination-traj

Objet créé : position

Effet : crée et renvoit un objet position qui représente la position qu'aurait le robot articulaire si celui-ci parcourait la trajectoire.

Valide-traj

Effet : teste la validité de la trajectoire. Une trajectoire est valide si le seul segment qu'elle comporte est valide. Active donc la compétence "valid-segm" du segment qui la constitue.

4.3.11. Compétences de la force

Valide-force

Effet : teste la validité de l'objet force. Une force est valide si les trois paramètres qui la composent sont des entiers et si

...

les deux premiers paramètres sont plus grand ou égal à 0 et plus petit ou égal à 7 et si le dernier est plus grand ou égal à 0 et plus petit ou égal à 99

Fermer

Effet : envoie un message au robot physique qui a pour effet de provoquer la fermeture de la pince avec une force donnée par l'instance cible du message.

Ouvrir

Effet : envoie un message au robot physique qui a pour effet de provoquer l'ouverture de la pince avec une force donnée par l'instance cible du message.

4.3.12. Compétences du temps

Valide-tps

Effet : teste la validité de l'objet temps. Le temps est valide s'il est représenté par un entier positif ou nul.

Attendre

Effet : envoie un message au robot physique qui a pour effet de bloquer son interpréteur de commande pendant un temps exprimé par l'objet cible du message.

4.3.13. Compétences du robot physique

Le robot physique est un concept appartenant à la couche inférieure à la couche articulaire. Nous n'en parlerons donc que très brièvement. Le robot physique que l'on désire programmer est aussi caractérisé par une série de compétences. Ces compétences sont "cablées" et correspondent au jeu d'instructions défini par le constructeur. Aussi le robot RM 501 est-il caractérisé par les compétences NT, PS, MO, MC, ...

L'annexe A de ce travail explique en détail un sous-ensemble du jeu d'instructions défini par le constructeur.

## 4.4. Objets cibles des messages envoyés par le robot articulaire

Parmi les objets manipulés dans la couche articulaire, un sous-ensemble seulement est connu du robot articulaire. C'est à ces objets que le robot articulaire va envoyer des messages pour actionner le robot physique.

Dressons simplement la liste de ces objets :

- la trajectoire
- le mouvement
- la vitesse
- la force
- la position
- le temps

L'implémentation des compétences et la structure des objets sont exposés dans l'annexe B de ce travail.

## 5. LA COUCHE "XYZ" ( Coiffet 1981, R.P. Paul 1981)

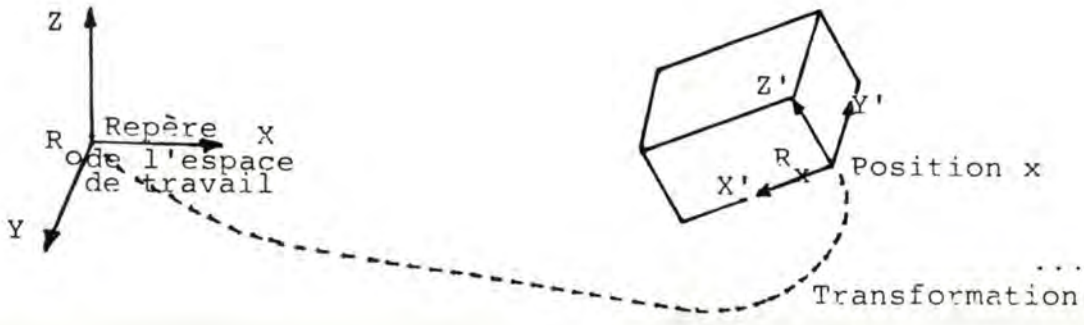
De par la nature des tâches à accomplir, les langages de programmation robotique doivent permettre la maîtrise parfaite des déplacements de l'organe terminal du robot. Dans la couche précédente, les mouvements étaient définis en terme de rotations des articulations (ou de translation des coulisses). Tout mouvement d'une articulation provoque dès lors un déplacement du corps qui lui est solidaire et de tous ceux situés en amont. Il est bien sûr possible de se limiter au niveau de la programmation de niveau articulaire et de définir les mouvements en terme de rotation des axes du robot. Cette approche qui est valable pour la programmation par apprentissage est très difficile à utiliser en programmation off-line car les positions ne sont plus déterminées par apprentissage mais par calcul.

Pour faciliter la programmation off-line, il est donc nécessaire de construire un outil qui permette de définir aisément les positions et orientations de la pince et des objets de l'espace de travail. Cet outil permettra par ailleurs d'assurer une indépendance de la programmation par rapport aux caractéristiques géométriques du robot (nombre de degrés de liberté, longueur des organes du bras et amplitude de rotation des articulations associées à chaque organe).

Le premier chapitre exposera les éléments théoriques nécessaires à l'élaboration d'un tel outil, le deuxième chapitre définira les concepts inhérents à la programmation du niveau XYZ spécifiques à ce travail.

### 5.1. Eléments théoriques

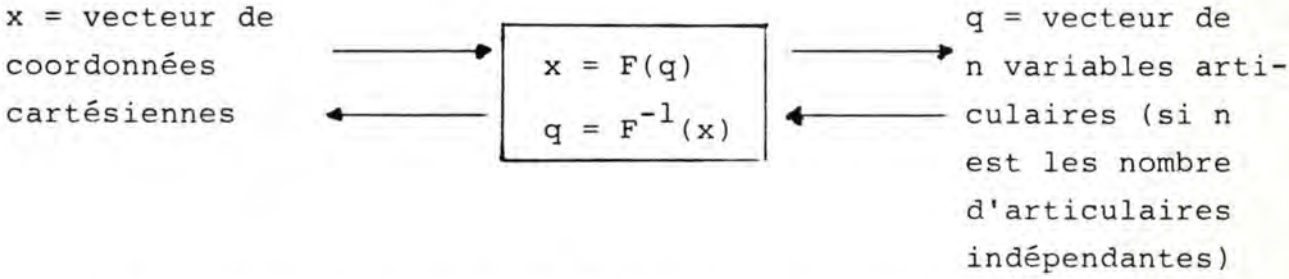
La méthode la plus naturelle permettant la définition de la position de la pince et des objets consiste à attribuer un repère cartésien $R_o$ à l'espace de travail du robot. Ce repère est très souvent lié à la base du robot. A toute position x, on attribue un repère ($R_x$). A l'aide d'une notation simple, on représente la transformation qui permet le "passage" du repère $R_o$ au repère $R_x$.



La définition des transformations fait l'objet du paragraphe 5.1.1.

Il n'est pas suffisant de pouvoir exprimer des positions, encore faut-il pouvoir les transformer en positions qui ont une signification pour le robot. Ce dernier ne peut atteindre des positions que si celles-ci sont décrites dans un code assimilable par l'unité de pilotage. Pour l'unité de pilotage, une position doit être définie à l'aide de coordonnées articulaires.

Il faudra donc créer un transformateur de coordonnées qui transforme les positions définies dans l'espace cartésien en "vecteurs" de coordonnées articulaires et vice-versa.




Les vecteurs x et q sont liés par une relation F non linéaire relativement complexe qui est déduite des caractéristiques géométriques et mécaniques du robot.

Nous aborderons au paragraphe 5.1.3. les problèmes de la modélisation mathématique du robot qui nous permettra de déterminer l'expression de la fonction F et de son inverse.

Au paragraphe 5.1.2., nous définirons un outil graphique qui permet de représenter de manière non ambiguë les caractéristiques géométriques et mécaniques d'un robot. Cet outil graphique nous sera bien utile car il permettra de représenter les robots pour lesquels on désire déterminer l'expression de F et de son inverse.

### 5.1.1. Les transformations dans l'espace de travail

Les transformations sont les mouvements nécessaires pour déplacer un objet d'une position cartésienne à une autre. La position d'un objet peut être considérée comme le résultat d'une transformation ayant pour origine le système d'axe de référence de l'espace de travail. Il existe deux types de transformations : la translation et la rotation.

...

5.1.1.1. La translation

La translation est une transformation qui permet d'exprimer le passage d'un repère $R_i$ à un repère $R_{i+1}$ par déplacement de l'origine de $R_i$ selon un vecteur $\vec{V}$ sans changement de son orientation.

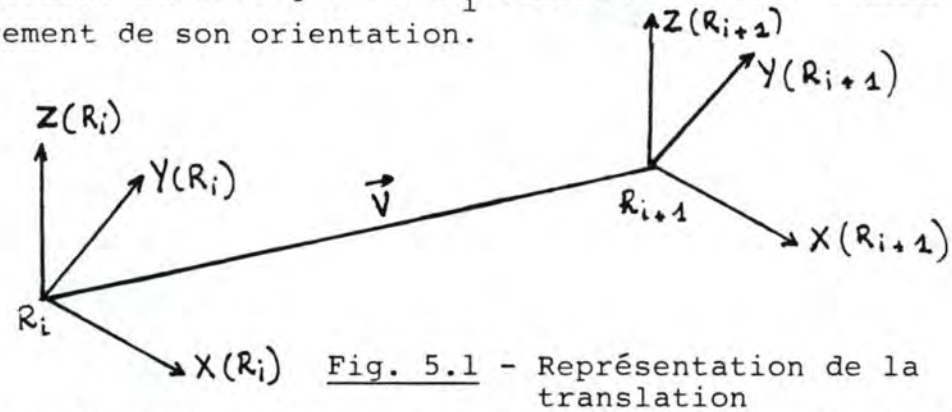



Fig. 5.1 - Représentation de la translation

La position cartésienne du repère $R_{i+1}$ sera le résultat de trois transformations :

- une translation de $V_x$ unités selon l'axe $X(R_i)$;
- une translation de $V_y$ unités selon l'axe $Y(R_i)$;
- une translation de $V_z$ unités selon l'axe $Z(R_i)$.

Si $V_x$, $V_y$ et $V_z$ sont les coordonnées de l'origine du repère $R_{i+1}$ dans $R_i$.

L'expression analytique des coordonnées dans $R_i$ d'un point P de coordonnées $X_{i+1}$, $Y_{i+1}$, $Z_{i+1}$ exprimées dans le repère $R_{i+1}$ sera de la forme :

$$P(R_i) = P(R_{i+1}) + (X_T, Y_T, Z_T)$$

si $X_T$, $Y_T$ et $Z_T$ sont les coordonnées de l'origine du repère $R_{i+1}$ dans le repère $R_i$ et si le passage d'un repère à l'autre est obtenu par translation.

La translation est le résultat d'une simple addition vectorielle.

5.1.1.2. La rotation

La rotation est une transformation qui exprime un changement de base. Les coordonnées de tout point p subissant une rotation restent inchangées par rapport à la nouvelle base, mais comme la nouvelle base est différente de l'ancienne le point p change de coordonnées par rapport à cette dernière.

...

Soit un repère $R_i$. On désire effectuer une rotation d'angle $\Theta$ autour de l'axe $X(R_i)$ du repère. La rotation du repère va donner lieu à un nouveau repère $R_{i+1}$ constitué des axes $X_{i+1}(R_{i+1})$, $Y_{i+1}(R_{i+1})$ et $Z_{i+1}(R_{i+1})$.

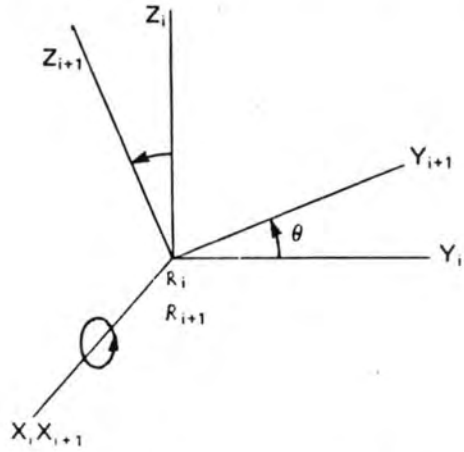



Fig. 5.2 - Représentation de la rotation

Rappelons l'expression générale d'une transformation de rotation d'un repère autour de ses axes :

Si $(X_{i+1}, Y_{i+1}, Z_{i+1})$ dénotent la position d'un point dans le repère $R_{i+1}$, la même position dans le repère $R_i$ est donnée par l'expression suivante :

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta \\ 0 & \sin\Theta & \cos\Theta \end{bmatrix}}_{\text{Rot } (x,\Theta)} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$$

La rotation d'un angle $\Theta$ autour de l'axe des X est donc entièrement caractérisée par la matrice $\text{Rot}(x,\Theta)$. De même, on aura :

$$\text{Rot } (y,\Theta) = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ -\sin\Theta & 0 & \cos\Theta \end{bmatrix}$$

...

et

$$\text{Rot } (z,\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Chaque ligne de ces matrices de rotation représente les coordonnées des projections des vecteurs directeurs du nouveau repère dans l'ancien.

La rotation inverse d'une rotation est caractérisée par l'inverse de la matrice associée, qui se réduit simplement dans ce cas à sa transposée. Cette propriété nous sera utile par la suite.

Par convention, le sens positif correspond au sens de rotation des aiguilles d'une montre.

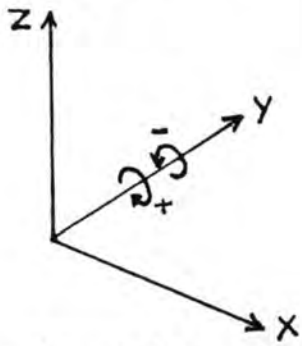



Fig. 5.3 - Convention du sens positif de rotation

5.1.1.3. Composition des transformations de rotation et de translation

Il suffit de composer ces deux types de transformations pour obtenir des transformations articulaires.

En toute généralité, un repère $R_{i+1}$ peut être obtenu par des rotations du repère $R_i$ autour de son axe $X_i, Y_i, Z_i$ d'un angle respectif de $\alpha, \beta$ et $\gamma$ et combinées par des translations de $V_x, V_y$ et $V_z$ unités respectivement selon ses axes $X_i, Y_i, Z_i$.

L'expression des coordonnées de tout point (x,y,z), exprimé dans le repère $R_{i+1}$, sera obtenue dans le repère $R_i$ à l'aide de la composition des transformations suivantes :

...

Nous représenterons par x'y'z' les coordonnées recherchées

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} + \text{Rot}\,(z,\gamma) * \text{Rot}\,(y,\beta) * \text{Rot}\,(x,\alpha) \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Ou encore :

$$\begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} + \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

En admettant que le repère $R_i$ ait subi une rotation d'angle $\alpha$ puis $\beta$ puis $\gamma$ respectivement autour de ses axes $X_i$, $Y_i$ et $Z_i$.

5.1.2. Représentation graphique des systèmes mécaniques articulés

(P. COIFFET 1981, LOPEZ & FAULC 1984)

Dans la suite du travail, nous serons amenés à déterminer le modèle mathématique d'un robot. L'obtention de ce modèle sera facilitée par l'utilisation d'une représentation graphique du robot étudié. Ce graphisme doit pouvoir représenter de façon claire les organes du SMA et leurs liaisons avec leurs prédecesseurs et successeurs.

On peut représenter les organes par de simples segments de droites car dans notre cas, seule leur longueur nous importe. Il reste encore le problème de la représentation des liaisons mécaniques entre les organes car cette représentation doit exprimer la mobilité d'un organe par rapport à un autre.

Il existe des normes AFNOR (Fig. 5.4) pour la représentation de ces liaisons mécaniques (normes NF E04-E015)

...

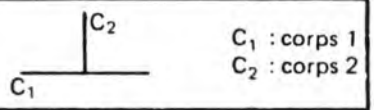
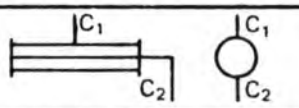
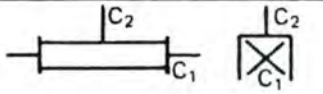
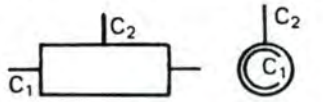
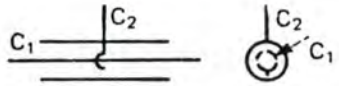
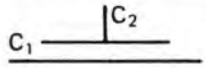
| Nom de la liaison | Mouvements relatifs | Nombre de d.d.l. | Symboles |
|---|---|---|---|
| Liaison encastrement | 0 rotation<br>0 translation | 0 |  |
| Liaison pivot | 1 rotation<br>0 translation | 1 |  |
| Liaison glissière | 0 rotation<br>1 translation | 1 |  |
| Liaison glissière hélicoïdale | 1 rotation<br>1 translation conjuguées | 1 |  |
| Liaison pivot glissant | 1 rotation<br>1 translation | 2 |  |
| Liaison appui plan | 1 rotation<br>2 translations | 3 |  |
| Liaison rotule | 3 rotations<br>0 translation | 3 |  |

Fig. 5.4 - Représentation normalisée de quelques liaisons

Ces normes peuvent être utilisées librement. On peut y ajouter des flèches et d'autres annotations qui rendent le graphisme plus "parlant".

La Fig. 5.4 reprend la liste des liaisons les plus courantes entre deux organes appelés C1 et C2 dans la figure. Ces liaisons y sont représentées de manière "stylisées" de profil et de face.

Passons en revue la liste des liaisons :

- la liaison encastrement : n'assure aucune mobilité entre les deux organes liés. Il s'agit typiquement de la liaison de la base du robot avec la table ou le sol sur lequel elle repose.

...

- la liaison pivot : cette liaison est certainement la plus courante : les deux organes sont liés par un axe de rotation. Cet axe peut être perpendiculaire à l'axe des organes ou dans leur prolongement.
- la liaison glissière : cette liaison est également fort courante : elle assure un mouvement de translation d'un des organes par rapport à l'autre. Les mécanismes téléscopiques et les mécanismes à crémaillère sont des cas typiques de liaison glissière.
- la liaison glissière hélicoïdale : cette liaison assure un mouvement de translation et de rotation entre les deux organes liés. Cette liaison me paraît peu commode car le mouvement de translation et de rotation sont toujours conjugués. La liaison d'un boulon avec un écrou est un cas typique de liaison glissière hélicoïdale.
- la liaison pivot glissant : cette liaison assure un mouvement de translation et de rotation entre deux organes liés. Cette fois-ci, les deux mouvements sont indépendants. Le déplacement d'une pièce tubulaire sur une tige lisse est un cas typique de cette liaison.
- la liaison appui plan : cette liaison assure des mouvements de translation d'un des organes par rapport à l'autre selon deux axes différents ainsi qu'un mouvement de rotation. Ces trois mouvements sont indépendants. Le robot SHAKEY, monté sur trois roulettes, est capable de se déplacer en X et en Y ainsi que de modifier son orientation. La liaison de la base de ce robot avec le sol est une liaison du type appui plan.
- la liaison rotule : est une liaison peu courante mais cependant fort pratique. Elle assure un mouvement de rotation de l'un des organes par rapport à l'autre autour de trois axes concourants. Le poignet de l'être humain est un cas typique de ce genre de liaison.

A titre d'exemple considérons la Fig. 5. page 5. Cette vue de profil nous donne une idée de la forme et de la longueur des organes du robot mais par contre ne nous donne aucune indication sur la nature des articulations qui lient les organes entre-eux.

...

Le même robot à l'aide de la norme AFNOR est représenté à la Fig. 5. :

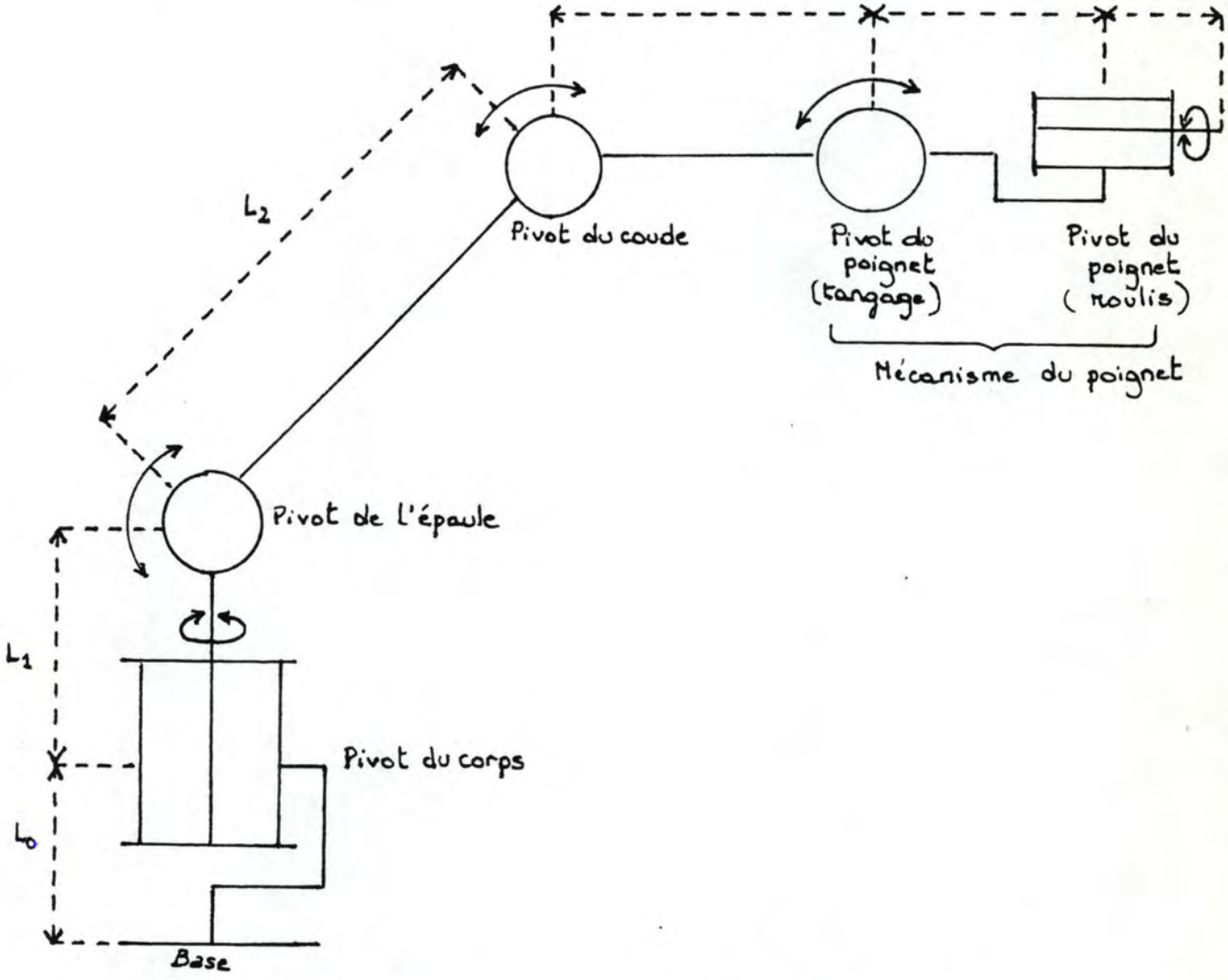



Fig. 5.5 - Représentation graphique normalisée du robot Mitsubishi movemaster RM 501.

On pourrait compléter ce schéma en donnant la masse de chaque organe ou encore l'amplitude maximale de chaque liaison pivot.

...

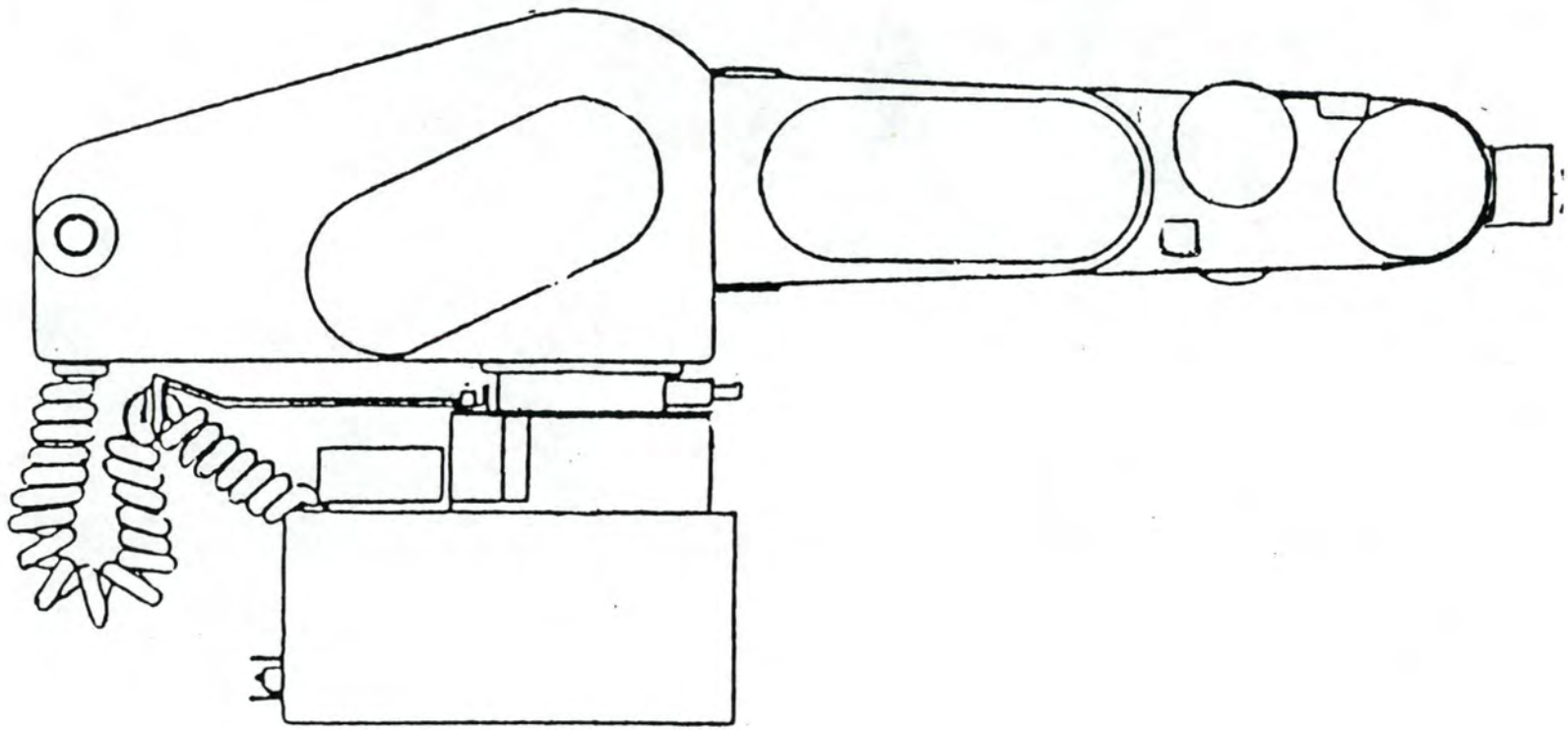


Fig. 5.6 - Projection horizontale du robot Movemaster RM 501

5.1.3. Modélisation mathématique du robot (P.COIFFET 1981)

L'utilisation d'un système mécanique articulé nous amène à étudier son comportement dans son espace de travail.

L'espace de travail est un espace cartésien à trois dimensions qui a pour référentiel un repère $R_o$ souvent lié à la base du robot.

La modélisation mathématique utilise des éléments de la cinématique des systèmes mécaniques articulés.

L'étude du comportement des systèmes mécaniques articulés nécessite la connaissance des paramètres du robot.

- Les paramètres structuraux : ces paramètres donnent la longueur des différents membres du SMA et leur succession dans la chaîne des organes qui le constituent Nous utiliserons à cet effet, le modèle géométrique présenté au paragraphe précédent.
- Les variables articulaires : on précise l'amplitude du mouvement qu'un actionneur peut exercer sur une articulation à partir d'une position 0. La position 0 varie en fonction du choix de la "home position".

Les éléments de la cinématique que nous allons aborder concernent :

- L'établissement des coordonnées des différents points du système mécanique dans l'espace des tâches en fonction de la valeur des variables articulaires. Ce domaine se nomme aussi la cinématique directe. Nous allons nous intéresser particulièrement à la position et à l'orientation de l'organe terminal du robot.
- Le calcul de la valeur des variables articulaires pour une position et une orientation données de l'organe terminal du robot dans l'espace des tâches. Ce domaine s'appelle la cinématique inverse.

...

5.1.3.1. La cinématique directe

Considérons un SMA constitué d'une chaîne d'organes reliés par des pivots et une glissière. On associe à chaque organe un repère. Ce repère est solidaire de l'articulation qui lie un organe à son prédecesseur dans la chaîne car un de ses axes est toujours dans le prolongement de l'axe de symétrie de ce corps.

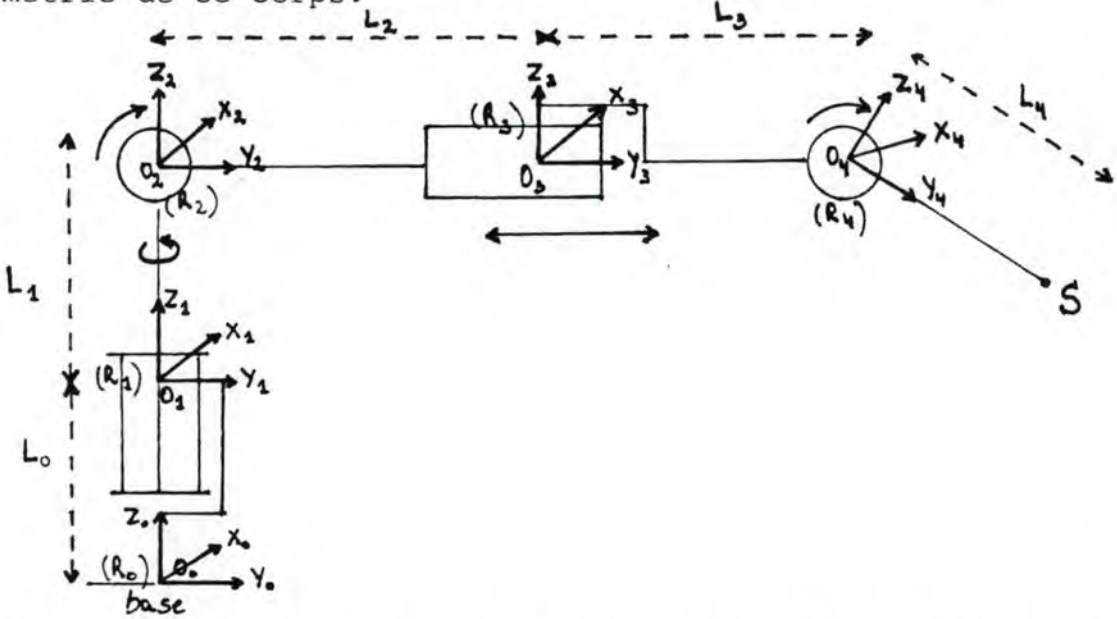
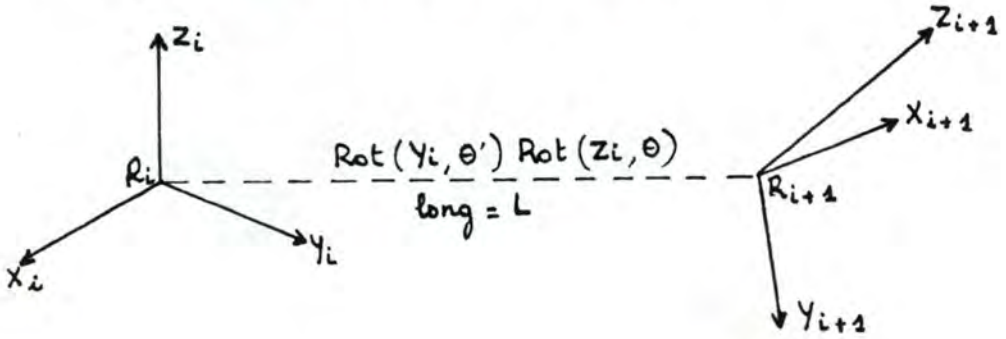



Fig. 5.7 - Représentation graphique d'un robot imaginaire muni de trois liaisons pivots et d'une glissière.

Théoriquement, pour passer du repère $R_i$ au repère $R_{i+1}$, on peut avoir jusqu'à six transformations :

- trois rotations autour de chaque axe du repère $R_i$ ;
- trois translations le long de chacun de ces mêmes axes.

S'il y a plus d'une transformation pour passer d'un repère à un autre, on insère entre ces deux repères des organes intermédiaires fictifs sans taille, ni masse, de manière à simplifier la démarche de calcul.




...

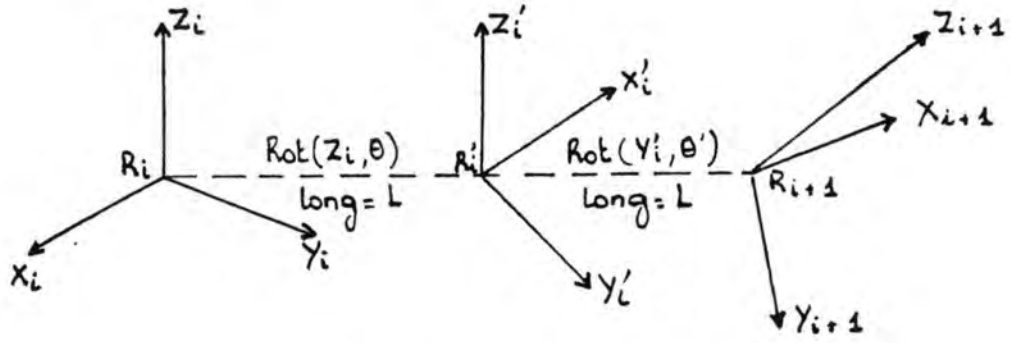



Fig. 5.8 - Insertion d'un organe intermédiaire fictif entre deux organes dont les repères associés sont séparés par deux rotations.

Dans l'exemple de la Fig. 5.8, chaque repère est séparé de son prédécesseur et de son successeur par une seule transformation de rotation. Pour simplifier davantage les calculs, il est préférable qu'un des axes de chaque repère soit dans les prolongement de l'axe de symétrie de l'organe auquel il est associé.

Nous observerons ces hypothèses simplificatrices par la suite.

5.1.3.1.1. Calcul de l'orientation d'un organe par rapport à un repère lié à un organe en aval de la chaîne

L'orientation d'un repère par rapport à un autre peut être considéréecomme le résultat de plusieurs opérations de rotation. Pour déterminer cette orientation, on ignore les translations qui séparent le repère $R_i$ du repère $R_{i+1}$ car elles n'ont aucune influence sur l'orientation des organes. La détermination de l'orientation d'un organe $C_{i+1}$ par rapport à un repère $R_i$, consiste à trouver la valeur des coordonnées des vecteurs unitaires du repère $R_{i+1}$ exprimées dans le repère $R_i$. Les repères associés à chacun des organes seront donc "agglomérés" à l'origine du repère $R_i$.

Pour passer du repère $R_i$ au repère $R_{i+1}$, supposons que l'on exécute une rotation $\theta$ du repère $R_i$ autour de son axe $X_i$. La transformation s'exprimera à l'aide de la transformation $Rot(X_i,\theta)$ définie précédemment et que l'on notera $M_i^{i+1}$.

...

Tout vecteur $\vec{V}$ exprimé dans le repère $R_{i+1}$ peut dès lors être exprimé dans le repère $R_i$ grâce à la relation

$$\vec{V}(R_i) = Rot(x_i,\theta)\ \vec{V}(R_{i+1})$$
$$= M_i^{i+1}\ \vec{V}(R_{i+1})$$

cette équation est récurrente. On a par exemple :

$$\vec{V}(R_{i-1}) = M_{i-1}^{i}\ M_i^{i+1}\ \vec{V}(R_{i+1})$$

d'une manière générale, on a donc :

$$\vec{V}(R_i) = M_i^{i+n}\ \vec{V}(R_{i+n}) \qquad \forall\ n \geqslant 0$$

avec $$M_i^{i+n} = M_i^{i+1}\ M_{i+1}^{i+2}\ \ldots\ M_{i+n-1}^{i+n}$$

5.1.3.1.2. Calcul de la position d'un organe de la chaîne par rapport au repère lié à un organe en aval

Pour dégager une formule, nous allons utiliser la Fig. 5.7 .

Supposons que l'on veuille calculer la position du point S du repère $R_4$ dans le repère $R_i$ lié à un organe en aval.

On peut écrire la relation vectorielle

$$\overrightarrow{O_iS} = \overrightarrow{O_iO_4} + \overrightarrow{O_4P}$$

Si on exprime cette relation dans le repère $R_i$, on obtient

$$\overrightarrow{O_iS}(R_i) = \overrightarrow{O_iO_4}(R_i) + \overrightarrow{O_4S}(R_i)$$

Le vecteur $\overrightarrow{O_4S}$ est connu dans le repère $R_4$. Pour le connaître dans le repère $R_i$, il suffit de lui appliquer la transformation $M_i^4$ définie précédemment.

La relation prendra alors la forme

$$\overrightarrow{O_iS}(R_i) = \overrightarrow{O_iO_4}(R_i) + M_i^4\overrightarrow{O_4S}(R_4) \qquad (i < 4)$$

A titre d'exemple, nous allons calculer la position du point S par rapport à la base du robot (repère $R_o$).

...

$$\overrightarrow{O_oS}(R_o) = \overrightarrow{O_oO_4}(R_o) + M_o^4 \overrightarrow{O_4S}(R_4)$$

$$\overrightarrow{O_oO_4}(R_o) = \overrightarrow{O_oO_3}(R_o) + M_o^3 \overrightarrow{O_3O_4}(R_3)$$

$$\overrightarrow{O_oO_3}(R_o) = \overrightarrow{O_oO_2}(R_o) + M_o^2 \overrightarrow{O_2O_3}(R_2)$$

$$\overrightarrow{O_oO_2}(R_o) = \overrightarrow{O_oO_1}(R_o) + M_o^1 \overrightarrow{O_1O_2}(R_1)$$

or

$$M_o^k = M_o^1 \cdot M_1^2 \cdots M_{k-1}^k \qquad (1 \leq k \leq 4)$$

et, de par les conventions sur la définition des repères,

$$\overrightarrow{O_4S}(R_4) = (O, L_4, O)^T$$

$$\overrightarrow{O_3O_4}(R_3) = (O, L_3, O)^T$$

$$\overrightarrow{O_2O_3}(R_2) = (O, L_2, O)^T$$

$$\overrightarrow{O_1O_2}(R_1) = (O, O, L_1)^T$$

$$\overrightarrow{O_oO_1}(R_o) = (O, O, L_o)^T$$

On obtient dès lors :

$$\overrightarrow{O_oS}(R_o) = \begin{bmatrix} O \\ O \\ L_o \end{bmatrix} + M_o^1 \begin{bmatrix} O \\ O \\ L_1 \end{bmatrix} + M_o^1 \cdot M_1^2 \begin{bmatrix} O \\ L_2 \\ O \end{bmatrix} + M_o^1 M_1^2 M_1^3 \begin{bmatrix} O \\ L_3 \\ O \end{bmatrix} + M_o^1 M_o^2 M_o^3 M_o^4 \begin{bmatrix} O \\ L_4 \\ O \end{bmatrix}$$

Remarques : - $M_2^3$ exprime l'orientation du repère $R_3$ par rapport au repère $R_2$. Comme, dans le cas qui nous occupe, ces deux repères ne sont séparés que par une translation, $M_2^3$ sera une matrice unitaire 3x3.

- $L_2$ n'est pas constante : cette longueur varie en fonction de la position de la coulisse liant l'organe C2 à l'organe C3.

...

5.1.3.1.3. Orientation de l'outil

L'outil est l'organe situé le plus en amont dans la chaîne des organes du SMA. Dans la majorité des cas et dans le cas qui nous occupe, cet organe est une pince contrôlée par un moteur.

L'orientation de l'outil est caractérisée par trois angles $\alpha$, $\beta$ et $\gamma$. On associe un trièdre $(X_e, Y_e, Z_e)$ à cet outil. L'axe $X_e$ se trouve sur l'axe de symétrie de l'outil, $Y_e$ sur son axe latéral et $Z_e$ sur son axe vertical. L'orientation de ce trièdre par rapport au trièdre de base $(X_o, Y_o, Z_o)$ de l'espace de travail exprime l'orientation de l'outil auquel il est rattaché.

Dans le cas de la pince, on associera ce trièdre à son centre de serrage situé à l'extrémité et à mi-distance des mâchoires.

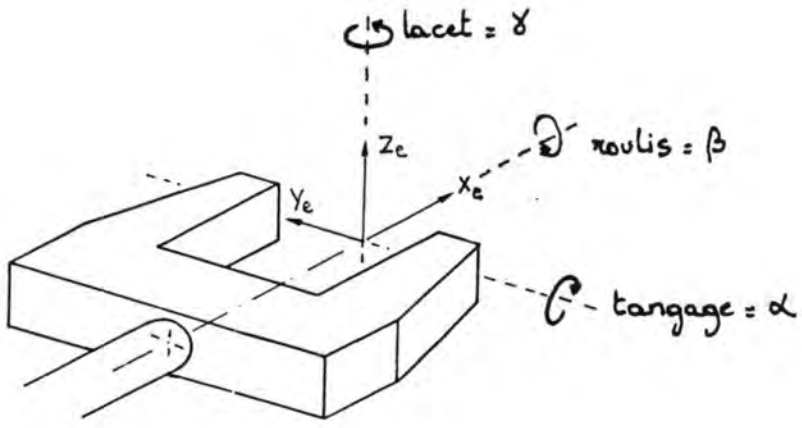



Fig. 5.9 - Orientation de l'outil

- L'angle $\alpha$ est l'angle formé par $X_e$ par rapport à $X_o$. Il est obtenu par rotation autour de l'axe $Y_e$. Cette rotation s'appelle le tangage.
- L'angle $\beta$ est l'angle formé par $Y_e$ par rapport à $Y_o$. Il est obtenu par rotation autour de l'axe $X_e$. Cette rotation s'appelle le roulis.
- L'angle $\gamma$ est l'angle formé par $X_e$ et $Y_e$ respectivement par rapport à $X_o$ et $Y_o$. Il est obtenu par rotation autour de $Z_e$. Cette rotation s'appelle le lacet.

...

5.1.3.1.4. Orientation du poignet

Le poignet est l'articulation qui relie la pince à l'avant bras. L'orientation du poignet est exprimée par un trièdre $T_p(X_p,Y_p,Z_p)$ par rapport au trièdre $T_a(X_a,Y_a,Z_a)$ solidaire de l'articulation de l'avant bras (coude). Le trièdre $T_p$ est solidaire de l'articulation du poignet, il est parallèle à $T_e$ et son axe $X_p$ se trouve sur l'axe de symétrie de la pince.

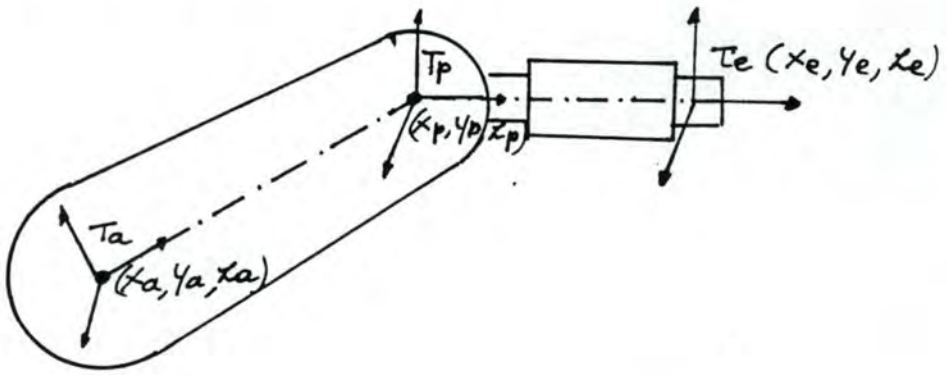



Fig. 5.10 - Orientation du poignet

Le poignet peut être animé à l'aide de trois mouvements :

- un mouvement rotatif autour de l'axe $Y_p$ nous donne l'inclinaison;
- un mouvement rotatif autour de l'axe $X_p$ nous donne la torsion;
- un mouvement rotatif autour de l'axe $Z_p$ nous donne la rotation (*).

L'orientation du poignet donne également l'orientation relative de la pince car les repères $T_e$ et $T_a$ sont parallèles. Cette orientation relative sera donc exprimée par rapport au repère $T_p$ de l'avant-bras.

(*) Je n'ai pas trouvé le nom de ce troisième mouvement, aussi l'ai-je baptisé simplement "rotation".

...

5.1.3.1.5. Application de la cinématique directe à un cas concret

Il s'agit de déterminer la position et l'orientation de l'organe terminal du robot RM 501 par rapport à un repère lié à sa base, connaissant la configuration de ses organes exprimée par un vecteur de coordonnées articulaires, autrement dit, on veut calculer la transformation

$$(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5) \text{ --------- } (x,y,z,\alpha,\beta,\gamma)$$

où x,y,z expriment la position de l'organe terminal et $\alpha,\beta,\gamma$ expriment son orientation.

Dans le cas qui nous occupe, on peut scinder le problème en deux sous-problèmes :

1° Détermination de la position de l'organe terminal dans l'espace de travail;

2° Détermination de l'orientation de l'outil.

1° Détermination de la position de l'organe terminal du robot dans l'espace de travail

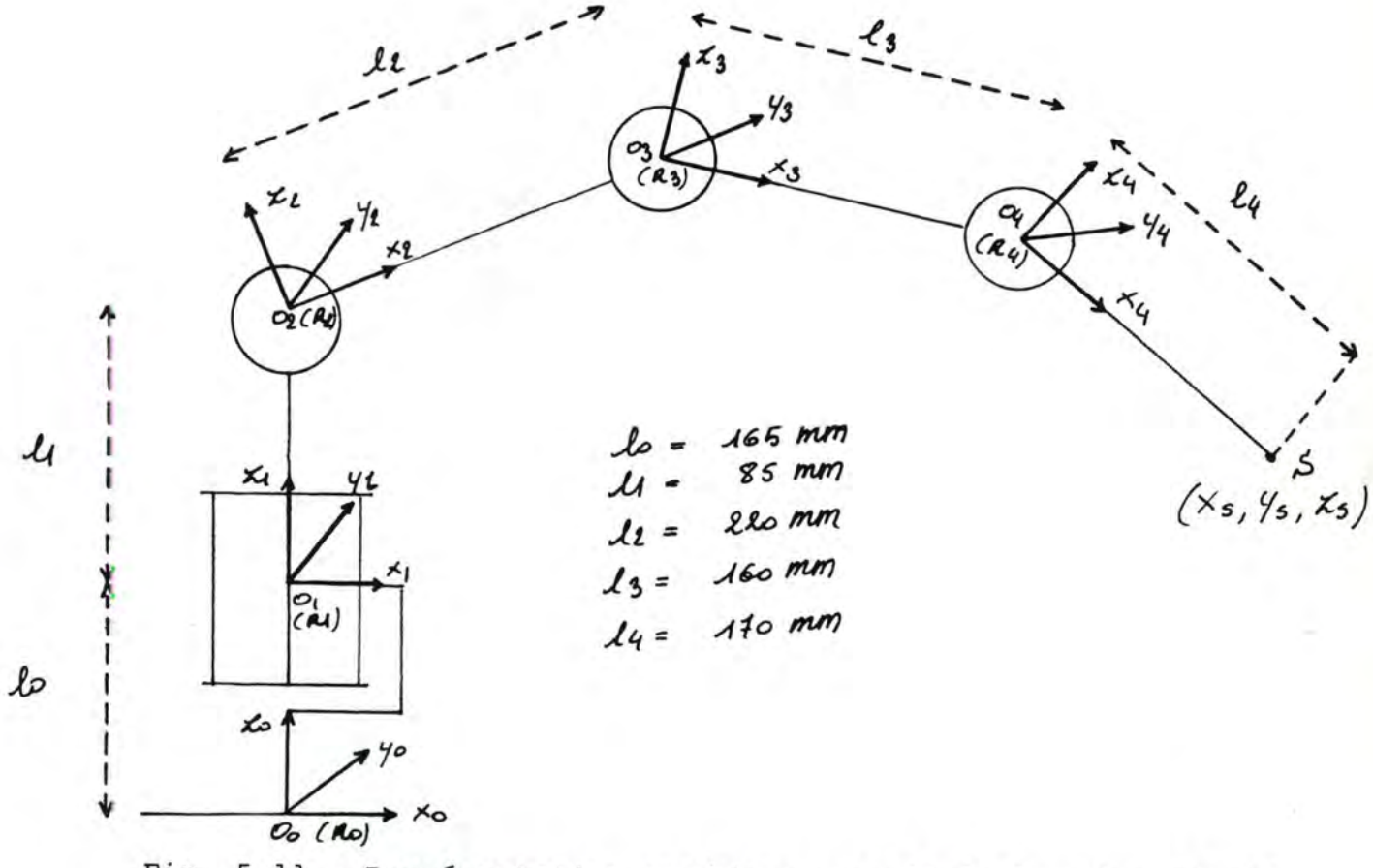



Fig. 5.11 - Représentation graphique normalisée du robot RM 501 sans son pivot de roulis

...

Remarque 1 : Les figures 5.5 et 5.11 représentent en fait le même robot. On remarque cependant que la liaison pivot la plus en amont de la figure 5.5 n'a pas été représentée dans la figure 5.11 . Ce pivot assure le mouvement de rotation de la pince autour de son axe de symétrie. Nous pouvons donc l'ignorer provisoirement car il n'y a aucune influence sur la position de l'extrémité de la pince par rapport au repère de l'espace de travail.

Remarque 2 : La longueur $L_4$ est la distance qui sépare l'articulation du poignet du centre de serrage de la pince situé à l'extrémité de ses mâchoires.

Nous pouvons appliquer la formule encadrée de la page 5.15 où les valeurs de $M_i^i$ sont les suivantes, dans le cas de notre robot.

$$M_o^1 = \mathrm{Rot}(Z_1, \theta_1)$$

$$M_1^2 = \mathrm{Rot}(Y_2, \theta_2)$$

$$M_2^3 = \mathrm{Rot}(Y_3, \theta_3)$$

$$M_3^4 = \mathrm{Rot}(Y_4, \theta_4)$$

$$M_o^2 = M_o^1 \cdot M_1^2$$

$$= \begin{bmatrix} C_1, & -S_1, & 0 \\ S_1, & C_1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \begin{bmatrix} C_2, & 0, & S_2 \\ 0, & 1, & 0 \\ -S_2, & 0, & C_2 \end{bmatrix}$$

$$= \begin{bmatrix} C_1C_2, & -S_1, & C_1S_2 \\ S_1C_2, & C_1, & S_1S_2 \\ -S_2, & 0, & C_2 \end{bmatrix}$$

avec $C_i = \cos \theta_i$

$S_i = \sin \theta_i$

...

$$M_o^3 = M_o^2 \cdot M_2^3$$

$$= \begin{bmatrix} C_1C_2, & -S_1, & C_1S_2 \\ S_1C_2, & C_1, & S_1S_2 \\ -S_2, & 0, & C_2 \end{bmatrix} \begin{bmatrix} C_3, & 0, & S_3 \\ 0, & 1, & 0 \\ -S_3, & 0, & C_3 \end{bmatrix}$$

$$= \begin{bmatrix} C_1C_2C_3 - C_1S_2S_3, & -S_1, & C_1C_2S_3 + C_1S_2C_3 \\ S_1C_2C_3 - S_1S_2S_3, & C_1, & S_1C_2S_3 + S_1S_2C_3 \\ -S_2C_3 - C_2S_3, & 0, & -S_2S_3 + C_2C_3 \end{bmatrix}$$

$$M_o^4 = M_o^3 \cdot M_3^4$$

$$= \begin{bmatrix} C_1C_2C_3 - C_1S_2S_3, & -S_1, & C_1C_2S_3 + C_1S_2C_3 \\ S_1C_2C_3 - S_1S_2S_3, & C_1, & S_1C_2S_3 + S_1S_2C_3 \\ -S_2C_3 - C_2S_3, & 0, & -S_2S_3 + C_2C_3 \end{bmatrix} \begin{bmatrix} C_4, & 0, & S_4 \\ 0, & 1, & 0 \\ -S_4, & 0, & C_4 \end{bmatrix}$$

$$= \begin{bmatrix} C_1C_2C_3C_4 - C_1S_2S_3C_4 - C_1C_2S_3S_4 - C_1S_2C_3S_4, & -S_1, \\ S_1C_2C_3C_4 - S_1S_2S_3C_4 - S_1C_2S_3S_4 - S_1S_2C_3S_4, & C_1, \\ S_2C_3C_4 - C_2S_3C_4 + S_2S_3S_4 - C_2C_3S_4, & 0, \end{bmatrix}$$

$$\begin{matrix} C_1C_2C_3S_4 - C_1S_2S_3S_4 + C_1C_2C_3C_4 + C_1S_2C_3C_4 \\ S_1C_2C_3S_4 - S_1S_2S_3S_4 + S_1C_2S_3C_4 + S_1S_2C_3C_4 \\ -S_2C_3S_4 - C_2S_3S_4 - S_2S_3C_4 + C_2C_3C_4 \end{matrix} \Bigg]$$

Nous pouvons dès lors calculer les différents termes du membre de droite de la formule encadrée en page 5.15 de ce travail, ce qui donne :

...

$$M_o^1(0,0,L_1)^T$$

$$= \begin{bmatrix} C_1, & -S_1, & 0 \\ S_1, & C_1, & 0 \\ 0, & 0, & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ L_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ L_1 \end{bmatrix}$$

$$M_o^2(L_2,0,0)^T$$

$$= \begin{bmatrix} C_1C_2, & -S_1, & C_1S_2 \\ S_1C_2, & C_1, & S_1S_2 \\ -S_2, & 0, & C_2 \end{bmatrix} \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} C_1C_2L_2 \\ S_1C_2L_2 \\ -S_2L_2 \end{bmatrix}$$

$$M_o^3(L_3,0,0)^T$$

$$= \begin{bmatrix} C_1C_2C_3 - C_1S_2S_3, & -S_1, & C_1C_2S_3 + C_1S_2C_3 \\ S_1C_2C_3 - S_1S_2S_3, & C_1, & S_1C_2S_3 + S_1S_2C_3 \\ -S_2C_3 - C_2S_3, & 0, & -S_2S_3 + C_2C_3 \end{bmatrix} \begin{bmatrix} L_3 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} C_1C_2C_3L_3 - C_1S_2S_3L_3 \\ S_1C_2C_3L_3 - S_1S_2S_3L_3 \\ -S_2C_3L_3 - C_2S_3L_3 \end{bmatrix}$$

$$M_o^4(L_4,0,0)^T$$

$$= \begin{bmatrix} C_1C_2C_3C_4 - C_1S_2S_3C_4 - C_1C_2S_3S_4 - C_1S_2C_3S_4, & -, & - \\ S_1C_2C_3C_4 - S_1S_2S_3C_4 - S_1C_2S_3S_4 - S_1S_2C_3S_4, & -, & - \\ S_2C_3C_4 - C_2S_3C_4 + S_2S_3S_4 - C_2C_3S_4, & -, & - \end{bmatrix} \begin{bmatrix} L_4 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} C_1C_2C_3C_4L_4 - C_1S_2S_3C_4L_4 - C_1C_2S_3L_4 - C_1S_2C_3S_4L_4 \\ S_1C_2C_3C_4L_4 - S_1S_2S_3C_4L_4 - S_1C_2S_3S_4L_4 - S_1S_2C_3S_4L_4 \\ S_2C_3C_4L_4 - C_2S_3C_4L_4 + S_2S_3S_4L_4 - C_2C_3S_4L_4 \end{bmatrix}$$

...

En sommant ces différents termes de la formule encadrée reprise à la page 5.15, on obtient en définitive :

$$X_s(R_o) = C_1C_2L_2 + C_1C_2C_3L_3 - C_1S_2S_3L_3 + C_1C_2C_3C_4L_4$$
$$C_1S_2S_3C_4L_4 - C_1C_2S_3S_4L_4 - C_1S_2C_3S_4L_4$$

$$Y_s(R_o) = S_1C_2L_2 + S_1C_2C_3L_3 - S_1S_2S_3L_3 + S_1C_2C_3C_4L_4$$
$$-S_1S_2S_3C_4L_4 - S_1C_2S_3S_4L_4 - S_1S_2C_3S_4L_4$$

$$Z_s(R_o) = -S_2L_2 - S_2C_3L_3 - C_2S_3L_3 + S_2C_3C_4L_4 -$$
$$C_2S_3C_4L_4 + S_2S_3S_4L_4 - C_2C_3S_4L_4 + L_o + L_1$$

2° Détermination de l'orientation de l'outil

L'orientation du poignet est donnée par les paramètres $\theta_4$ et $\theta_5$ du vecteur donnant la position du robot dans l'espace des variables articulaires :

$\theta_4$ représente l'inclinaison du poignet;
$\theta_5$ représente la torsion du poignet.

Connaissant l'orientation relative de la pince, il est possible d'en déterminer l'orientation absolue.

Calcul du roulis

Le roulis reste inchangé lorsqu'il est exprimé par rapport au poignet ou par rapport au repère de l'espace de travail car tous les autres axes de rotation lui sont perpendiculaires.

$$\text{Roulis} = \theta_5 = \text{torsion.}$$

Calcul du tangage

Le tangage de l'organe terminal est fonction de l'orientation de l'épaule, du coude et de l'inclinaison du poignet car les trois axes qui contrôlent ces orientations sont tous parallèles.

L'angle de rotation de l'épaule = $\theta_2$
L'angle de rotation du coude = $\theta_3$
L'angle de l'inclinaison = $\theta_4$

donc,

...

tangage = $\theta_2 + \theta_3 + \theta_4$

Calcul du lacet

Le robot RM 501 est dépourvu d'axe de lacet. Cela ne veut pas dire pour autant que la pince est dépourvue du mouvement rotatif autour de son repère $Z_e$. On constate en effet que ce mouvement rotatif est fonction de la position de la pince dans l'espace de travail.

L'angle de ce mouvement rotatif est égal à la valeur de la variable articulaire exprimant l'angle du corps. L'angle de rotation des autres articulations n'a pas d'influence sur l'angle de lacet car leurs axes sont tous perpendiculaires à l'axe de rotation du corps.

donc,

Lacet = $\theta_1$.

5.1.3.1.6. Réalisation d'un module de calcul des éléments de la cinématique directe du RM 501

Le module de calcul est écrit en Prolog et est spécialement écrit pour le robot Movemaster RM 501 de Mitsubishi. Comme les calculs se font dans l'espace des réels, l'interpréteur Prolog, qui originalement ne connaît pas le type entier, a été étendu. Les extensions du langage et le module de la cinématique directe figurent en annexe C de ce volume.

5.1.3.1.7. Détermination de la cinématique du robot RM 501 à l'aide d'une méthode "sur mesure"

En observant les caractéristiques géométriques du robot, on peut trouver une méthode "sur mesure", plus simple, permettant de calculer la position et l'orientation de l'organe terminal par rapport à un repère lié à la base du robot.

J'ai donc inventé une méthode "sur mesure" qui a le mérite d'être plus simple et d'engendrer par conséquent moins de calculs que la précédente. Dans cette méthode, j'associe un vecteur à chaque organe orienté vers l'amont et situé sur son axe de symétrie. On peut déterminer la

...

position de l'extrémité de la pince dans l'espace de travail en sommant ces vecteurs.

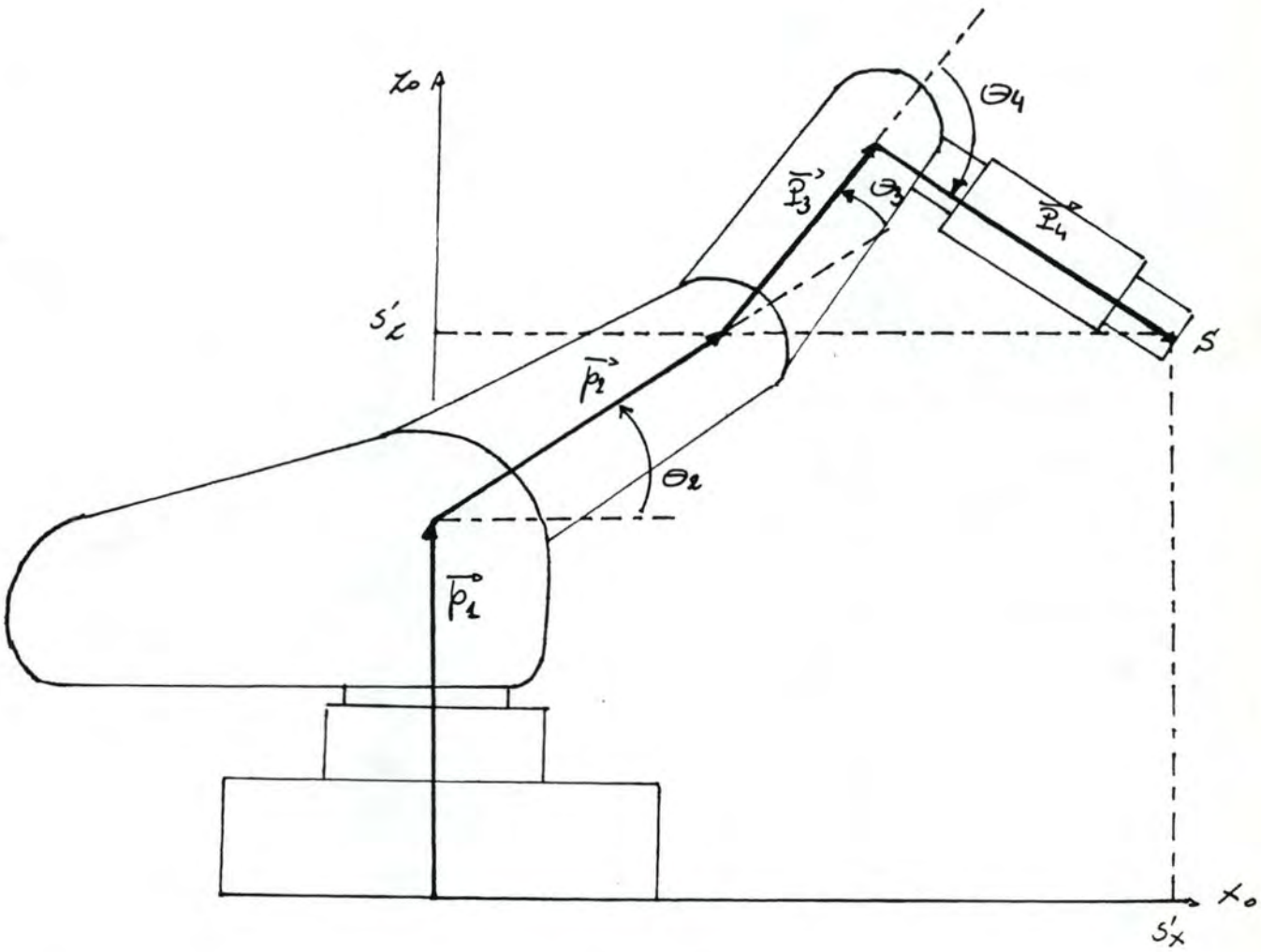



Fig. 5. 12 - Une méthode plus simple pour la détermination de la cinématique du robot RM 501.

La méthode se déroulera en trois temps :

1° On calcule une position provisoire de l'extrémité de la pince dans l'espace de travail en considérant porvisoirement que l'angle de rotation du corps est nul.

...

2° On calcule ensuite la véritable position de la pince dans l'espace de travail en tenant compte de l'angle de rotation du corps.

3° On détermine enfin l'orientation de la pince.

1° Calcul de la position provisoire de la pince

On notera $p_i^*$ les coordonnées des extrémités de tout vecteur $\overrightarrow{p_i}$ dont l'extrémité initiale a été translatée à l'origine du repère $R_o$ de l'espace de travail. On considère que l'angle du corps est provisoirement nul.

Appelons $S'(X'_s, Y'_s, Z'_s)$ la position provisoire de la pince dans l'espace de travail.

Le calcul de cette position sera obtenue par

$$S = \sum_i p_i^*$$

avec

$$p_1^* = (0,\ 0,\ L_o + L_1)$$

$$p_2^* = (\cos(\theta_2).L_2,\ 0,\ \sin(\theta_2).L_2)$$

$$p_3^* = (\cos(\theta_2 + \theta_3).L_3,\ 0,\ \sin(\theta_2 + \theta_3).L_3)$$

$$p_4^* = (\cos(\theta_2 + \theta_3 + \theta_4).L_4,\ 0,\ \sin(\theta_2 + \theta_3 + \theta_4).L_4)$$

donc,

$$X'_s = \cos(\theta_2).L_2 + \cos(\theta_2 + \theta_3).L_3 + \cos(\theta_2 + \theta_3 + \theta_4).L_4$$

$$Y'_s = 0$$

$$Z'_s = \sin(\theta_2).L_2 + \sin(\theta_2 + \theta_3).L_3 + \sin(\theta_2 + \theta_3 + \theta_4).L_4$$

2° Calcul de la véritable position de la pince

Appelons cette position $S(X_s, Y_s, Z_s)$.

Quel que soit l'angle $\theta_1$ du corps, la hauteur reste inchangée. Il nous restera encore à calculer la véritable valeur de l'abscisse et de l'ordonnée du point S.

$$X_s = \cos(\theta_1)\ .\ X'_s$$

$$Y_s = \sin(\theta_1)\ .\ Y'_s$$

$$Z_s = Z'_s$$

...

3° Calcul de l'orientation de la pince

Nous avons déjà abordé ce problème antérieurement (voir page 5.24).

tangage = $\theta_2 + \theta_3 + \theta_4$

roulis = $\theta_5$

lacet = $\theta_1$

5.1.3.1.8. Réalisation d'un module d'implémentation de la méthode "sur mesure"

Voir Annexe C.

5.1.3.2. Eléments de la cinématique inverse du robot (Coiffet 1981)

Supposons qu'un utilisateur désire déplacer un cylindre A pour le placer au point p de l'espace de travail. L'utilisateur connaît certaines grandeurs : dimension, coordonnées cartésiennes, orientation de l'objet A, exprimées par rapport au repère $R_o$ de l'espace de travail. Le problème posé consiste à déterminer la valeur à donner aux différentes variables articulaires de manière à amener le robot de la configuration initiale (saisir l'objet A) à la configuration finale (amener l'objet A au point p).

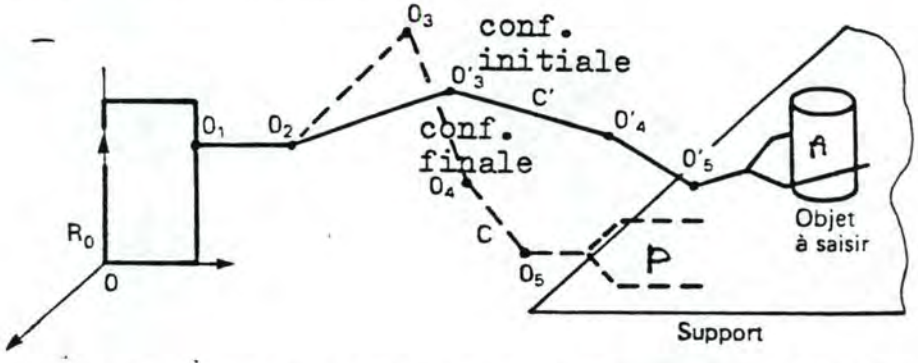



Fig. 5.13 - Configurations initiale et finale pour le déplacement d'un objet A à la position cartésienne p.

Le calcul des valeurs des différentes variables articulaires nécessite la connaissance :

- des paramètres structuraux du robot : limite de variation des articulations du robot, longueur des membres, etc...

...

- des positions et orientationsde l'organe terminal du robot exprimées dans l'espace de travail.
- de certaines contraintes spatiales du volume dans lequel s'effectue la tâche.

A partir de connaissances sur le robot et sur l'espace de travail, on transformera des coordonnées cartésiennes exprimées dans l'espace de travail (dimension 3) en coordonnées exprimées dans l'espace des variables articulaires (de dimension maximum 6)

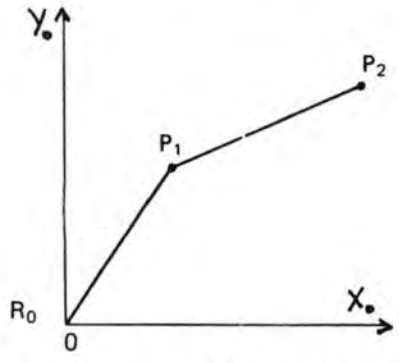
5.1.3.2.1. Notion de résolvabilité

Un manipulateur se dit résolvable s'il existe au moins un algorithme permettant de déterminer à partir d'une position et d'un orientation de l'organe terminal, les différentes valeurs articulaires correspondant à une configuration du manipulateur compatibles avec les contraintes imposées.

Plusieurs cas sont à envisager.

1) Absence de solution

a) Pour une raison d'ordre géométrique : il y a une incompatibilité entre la position à atteindre et les contraintes géométriques. Ce cas est illustré par la figure ci-dessous.
Considérons un mécanisme à deux degrés de liberté




$L_1 = dist(0,P)$

$L_2 = dist(P_1,P_2)$

Fig. 5.14 - Illustration de l'absence de solution avec un manipulateur à deux organes.

si on impose pour le point $p_2$ $p_{2x} = 0$

$p_{2y} = 0$

il est évident que le point (0,0) est hors de portée du mécanisme car $L_1 \neq L_2$

...

b) Pour une raison d'ordre mécanique

Les valeurs des variables articulaires ont une variation limitée inférieurement et supérieurement.

Ces limites sont dues à des contraintes mécaniques telles que :

- la présence d'une butée de fin de course;
- le couplage possible entre variables articulaires provenant du choix du mode de transmission entre les actionneurs et les articulations.

c) Pour une raison d'ordre mathématique

Dans les chapitres 5.1.3.1.2. et 5.1.3.1.3., nous avons dégagé l'expression générale d'une fonction permettant de calculer la position et l'orientation de l'organe terminal. Nous avions donc :

$$F(\theta_1\theta_2 \ldots\ldots \theta_n) \longrightarrow p(R_o)$$

n=nombre d'articulations indépendantes.

Si p est le point que l'on désire atteindre, et s'il n'est pas en contradiction avec les contraintes mécaniques et géométriques, il existe alors une valeur pour chaque variable articulaire satisfaisant l'objectif à atteindre (point p). On procède par inversion de la fonction F

$$F^{-1}\ (p(R_o)) \longrightarrow (\theta_1\theta_2 \ldots\ldots \theta_n)$$

Plusieurs problèmes peuvent se poser

- La fonction F est parfois difficile ou impossible à inverser. Pour en être convaincu, il suffit de regarder la forme des équations de la cinématique directe du robot RM 501.
- Si on arrive à inverser la fonction F, il reste encore à résoudre un système d'équations non linéaires pour déterminer la valeur des $\theta$. Cette difficulté croît avec le nombre d'équations couplées. On comprend alors aisément pourquoi il est possible de ne pas trouver de solution analytique alors qu'il existe une configuration du manipulateur qui satisfait l'objectif et qui est en accord avec les contraintes géométriques et mécaniques.

...

- Enfin, remarquons que si l'inversion est possible en théorie, elle peut être insatisfaisante en pratique car le temps de calcul nécessaire pour la transformation inverse est trop considérable. Le module de calcul de la transformation inverse est de loin le module le plus utilisé dans un environnement de programmation de niveau "XYZ". Si ce module est inefficace, il constituera, bien sûr, un très sérieux goulet d'étranglement pour l'environnement de programmation, remettant aussi son efficacité en question.

2) Infinité de solutions

Si le nombre K de contraintes (en particulier les positions et l'orientation de l'organe terminal) est inférieur au nombre N de degrés de liberté, lors de la résolution du système d'équations, il existe N-K inconnues qui peuvent prendre une valeur arbitraire sans pour autant être en contradiction avec les contraintes.

A titre d'exemple, considérons le porteur ci-dessous

contraintes $X_C = 0$

$Y_C = L_1 + L_2$

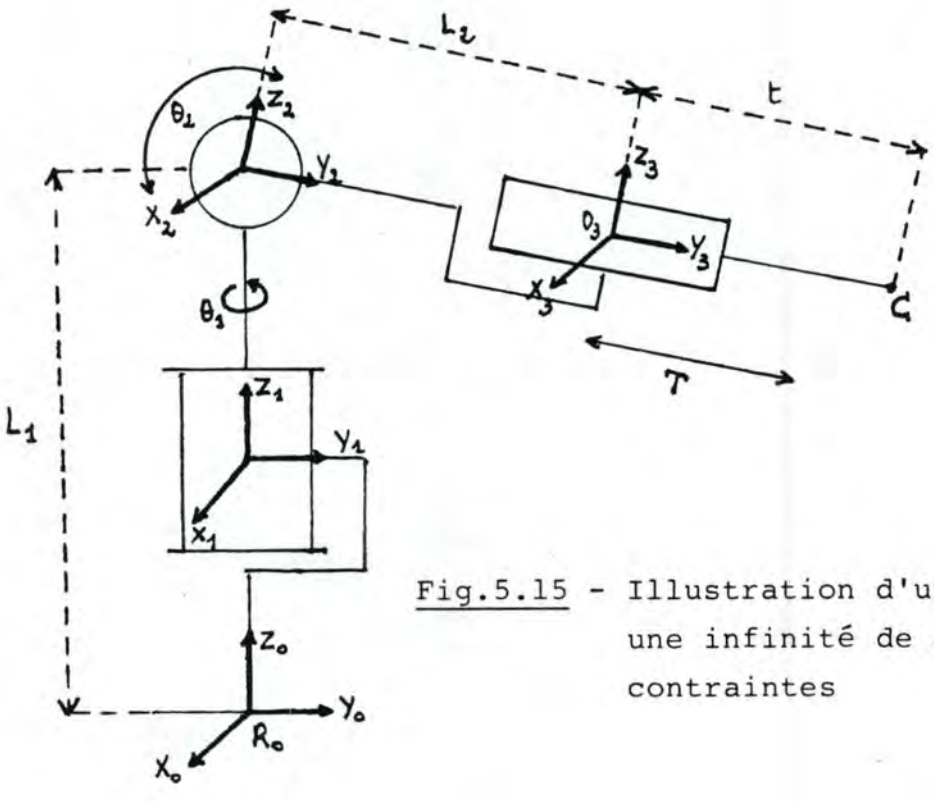



Fig.5.15 - Illustration d'un cas caractérisé par une infinité de solutions sous certaines contraintes

...

$$M_o^1 = \begin{bmatrix} C_1 & -S_1 & 0 \\ S_1 & C_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad M_1^2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_2 & -S_2 \\ 0 & S_2 & C_2 \end{bmatrix} \quad M_2^3 = \text{matrice unité } 3 \times 3$$

$$\overline{OC}(R_o) = \overline{OO}_1(R_o) + M_o^2\overline{O_1O}_2(R_2) + M_o^3\overline{O_3C}(R_3)$$

d'où

$$X_c(R_o) = -S_1C_2(L_2 + T)$$

$$Y_c(R_o) = C_1C_2(L_2 + T)$$

$$Z_c(R_o) = L_1 + S_2(L_2 + T)$$

Le système a pour solution

$$\theta_1 = 0$$

$$C_2(L_2 + T) = L_1 + L_2 \qquad C_i = \cos(\theta_i)$$

ou

$$\theta_1 = 0$$

$$C_2(L_2 + T) = -L_1 + L_2$$

d'où une infinité double de solutions en $\theta_2$ et T.

5.1.3.2.2. Résolvabilité du robot RM 501

On a déjà entrevu une manière de répondre au problème de la cinématique inverse du robot RM 501. On peut en effet inverser les équations cinématiques directes pour obtenir $F^{-1}$ $(p(R_o) : p(R_o) \longrightarrow (\theta_1\theta_2\theta_3\theta_4\theta_5)$. Pour obtenir les $\theta_i$, il faut résoudre un système d'équations non linéaires, ce qui provoque un grand nombre de calculs pour ne pas aboutir nécessairement à une solution.

Il faut essayer de résoudre le problème de la cinématique inverse à l'aide d'une méthode qui tient compte des particularités du système mécanique articulé. En observant les caractéristiques géométriques du bras articulé, on peut trouver une méthode particularisée qui nous permet de déterminer séquentiellement la valeur des variables arti-

...

culaires. J'ai donc scindé le problème de la cinématique du robot RM 501 en une suite de sous-problèmes plus simples. Chaque sous-problème permet le calcul de la valeur d'une ou de plusieurs variables articulaires indépendamment des autres ou permet simplement de simplifier le modèle géométrique du robot.

Nous déterminerons donc séquentiellement :

- La valeur de la variable articulaire $\Theta_1$. Cette valeur représentera l'angle de rotation du corps;
- Une simplification du modèle du robot par projection des traces de ses membres sur un plan. Cette projection nous permettra de raisonner dans un espace de travail à deux dimensions et de simplifier considérablement le calcul;
- La définition d'un repère de base de l'espace de travail à deux dimensions;
- La position de la pince dans cet espace de travail à deux dimensions;
- La définition de la position du poignet dans le repère de l'espace de travail à deux dimensions. La connaissance de cette position permettra une nouvelle simplification du modèle du robot;
- La valeur de la variable articulaire $\Theta_3$. Cette valeur représentera l'angle de rotation du coude;
- La valeur de la variable articulaire $\Theta_2$. Cette valeur représentera l'angle de rotation de l'épaule;
- L'orientation du poignet.

Sous-problème 1 : Calcul de l'angle $\Theta_1$ de rotation du corps

L'angle de rotation du corps du robot par rapport au repère X,Y,Z de l'espace de travail nous donnera la valeur $\Theta_1$. Le modèle géométrique du robot nous indique que les axes de rotation des pivots de l'épaule, du coude et du poignet sont parallèles. Par conséquent, les traces de la projection verticale de l'arrière-bras, de l'avant-bras et de la pince seront toujours dans un même axe. Nous pouvons déduire aisément la valeur $\Theta_1$ en raisonnant sur ces traces.

...

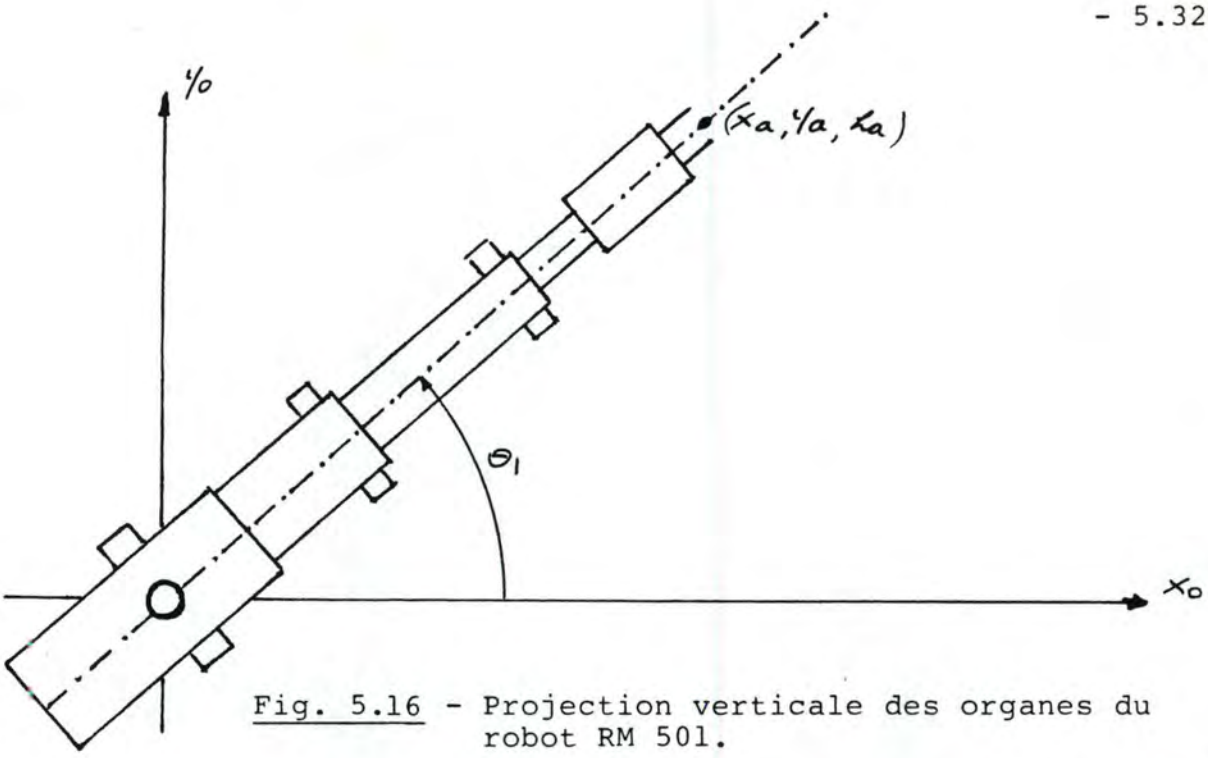



Fig. 5.16 - Projection verticale des organes du robot RM 501.

L'axe commun à l'arrière-bras, l'avant-bras et le poignet est une droite D passant par les points o et p qui ont respectivement pour coordonnées (o,o) et (x',y').

L'équation générale d'une droite dans $R^2$ en passant par 2 points quelconques $(X_a,Y_a)$, $(X_b,Y_b)$ est donnée par la formule

$$(Y - Y_b) = \frac{Y_b - Y_a}{X_b - X_a} \quad (X - X_b)$$

en substituant b par o et a par p, on obtient comme équation pour P

$$Y = \frac{Y_a}{X_a} \cdot X$$

Le rapport $\frac{Y_a}{X_a}$ désigne le coefficient de direction de la droite D. Le coefficient de direction est aussi l'expression de la tangente de l'angle formé par la droite D et l'axe X. Or, cet angle correspond à la valeur de la variable $\theta_1$ que l'on recherche. Nous avons donc :

$$\frac{Y_a}{X_a} = tg(\theta_1)$$

$$\theta_1 = arctg(\frac{Y_a}{X_a}) + k\pi$$

...

Sous-problème 2 : Simplification du modèle du robot par projection des traces de ses membres sur un plan

Pour une valeur $\theta_1$ donnée, le mouvement de la pince résultant de la variation de la valeur des variables articulaires $\theta_2, \theta_3, \theta_4$, se situe toujours dans un même plan vertical P. Ce plan P contient les axes de symétrie du corps, de l'avant-bras, de l'arrière-bras et de la pince. On peut donc réduire le problème posé dans un espace à deux dimensions en raisonnant sur les traces des membres du robot sur le plan P. Ces traces sont obtenues par projection horizontale et perpendiculaire au plan P sur ce même plan P. On obtient alors la représentation simplifiée du robot (Fig.5.11 ) où l'axe de rotation du corps a été supprimé. Il reste encore à définir une base B pour ce nouvel espace de travail à deux dimensions et à calculer la position de la pince dans cet espace.

Sous-problème 3 : Définition du repère de l'espace de travail à deux dimensions

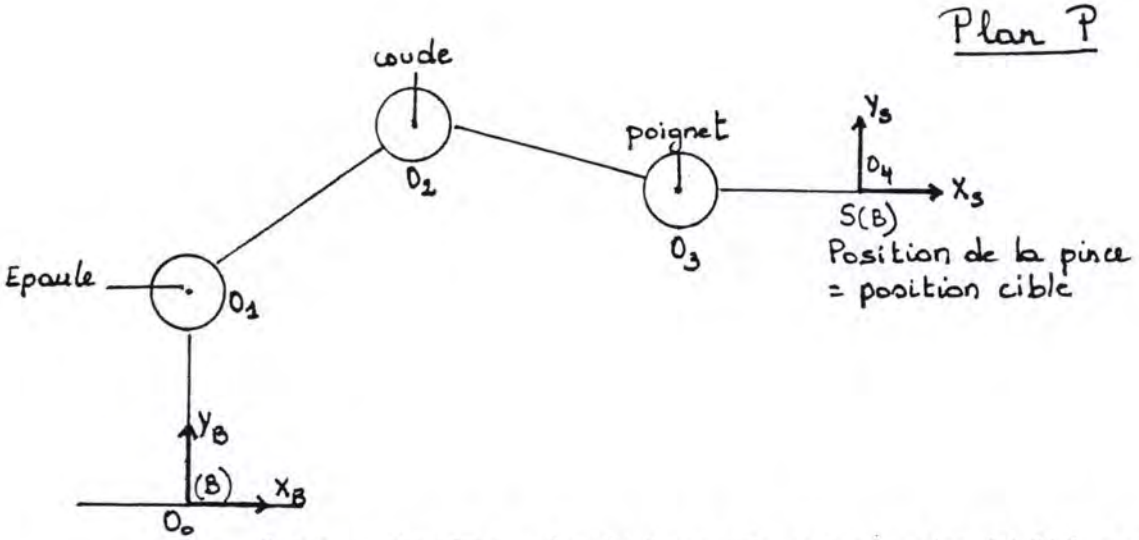



Fig. 5.17 - Modèle géométique du robot simplifié par la suppression de l'articulation du corps.

Les deux vecteurs du nouveau repère B doivent se situer dans le plan P. Appelons les $X_b$ et $Y_b$ les axes directeurs de ce repère.

...

- l'axe $X_b$ se situera sur la droite d'intersection du plan P avec le plan horizontal contenant les axes $X_o$ et $Y_o$ du repère $R_o$.
- l'axe $Y_b$ est identique à l'axe $Z_o$ du repère $R_o$.

Sous-problème 4 : Calcul du point S dans le repère B

Le point S représente l'extrémité de la pince. On connaît les coordonnées $(X_s, Y_s, Z_s)$ du point S dans le repère $R_o$. On peut donc calculer ses coordonnées $(X'_s, Y'_s)$ dans le repère B.

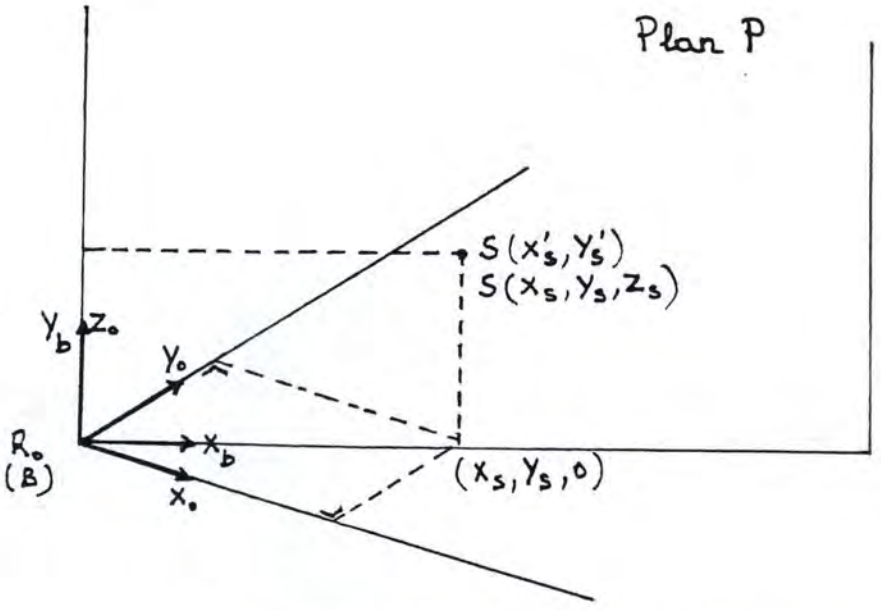



Fig. 5.18 - Coordonnées de l'extrémité de la pince dans le repère B d'un espace à deux dimensions.

* abscisse de S(B)

$X'_s$ = norme du vecteur d'extrémité $(X_s, Y_s, O)$

$$= \sqrt{X_s^2 + Y_s^2}$$

* hauteur de S(B)

La hauteur du point S dans la base B est donnée par $Y'_s$. Cette hauteur est identique à celle exprimée par la composante $Z_s$ des coordonnées de S dans $R_o$ car la projection horizontale sur le plan vertical P conserve la hauteur.

$$Y'_s = Z_s$$

...

Sous-problème 5 : Calcul de la position du poignet dans le repère B

La position du poignet est aussi la position de l'extrémité de l'avant-bras. Si on détermine cette position, on peut simplifier le modèle du robot en y supprimant le poignet et la pince. On obtiendra ainsi la figure 5. .

La position de l'extrémité de la pince est caractérisée par trois angles exprimant l'orientation du repère d'origine S. Ces angles sont le tangage, le roulis et le lacet.

L'angle du roulis n'a pas d'influence sur la position du poignet car l'axe du roulis est dans le prolongement de l'axe de symétrie de la pince.

L'angle du lacet a été supprimé par la projection de la pince sur le plan P. Cet angle n'aura pas d'influence sur la position du poignet dans l'espace de travail à deux dimensions de repère B.

L'angle de tangage de l'extrémité de la pince a, par contre, une influence sur la position du poignet dans l'espace de travail à deux dimensions de repère B.

On obtiendra la position du poignet par le raisonnement suivant :

Soit - $p(X_p, Y_p)$ la position du poignet
- ß, l'angle du tangage
- $S(X_s, Y_s)$ la position de l'extrémité de la pince exprimée par rapport au repère B.
- $L_5$ la distance qui sépare S de p
- $B_1$ le repère associé à S et qui définit l'orientation de la pince par rapport à B

...

On connaît la position de p exprimée dans le repère $B_1$ (Fig. 5. )

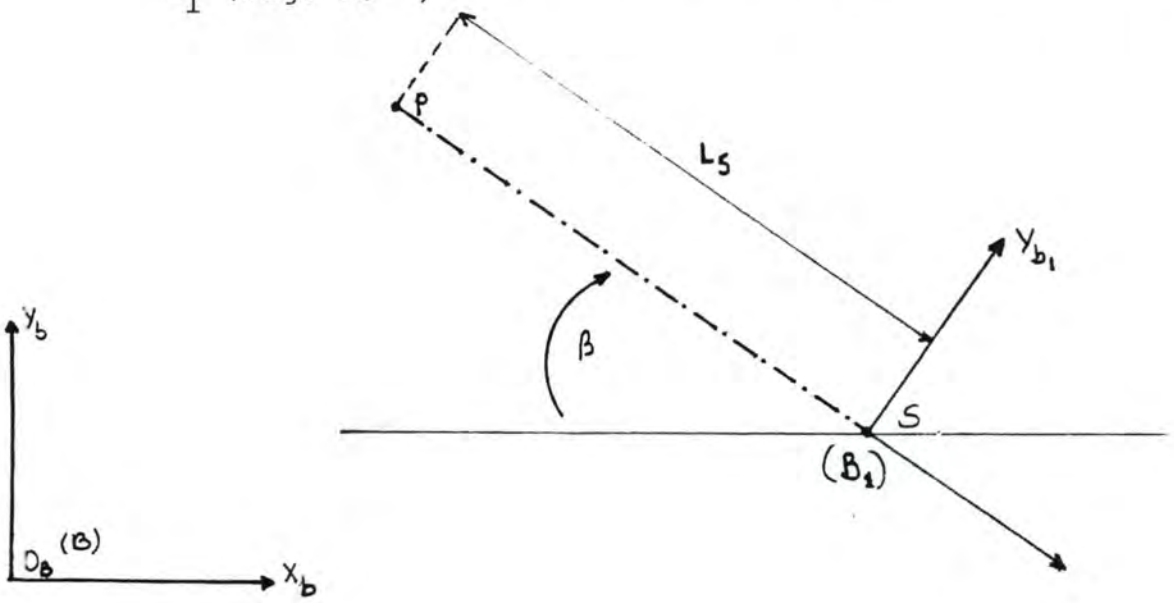



Fig. 5.19 - Influence du tangage de la pince sur la position du poignet.

Comme p et S sont distants d'une longueur $L_5$ et comme p se trouve dans le prolongement de l'axe $X_{B1}$, ses coordonnées dans $B_1$ seront $p(B_1) = (-L_5, 0)$. Il reste encore à trouver p(B). On utilisera la méthode exposée précédemment.

$$p(B) = S(B) + p(B_1)$$

$$p(B) = S(B) + M_{B1}^{B}\, p(B)$$

$$p(B) = (X_s, Y_s)^T + M_{B1}^{B}(-L_5, 0)^T$$

avec

$$M_{B1}^{B} = \begin{bmatrix} \cos\beta & \sin\beta \\ -\sin\beta & \cos\beta \end{bmatrix}$$

on obtient donc

$$X_p = X_s - L_5 \cos\beta$$

$$Y_p = Y_s + L_5 \sin\beta$$

Sous-problème 6 : Calcul de l'angle $\theta_3$ de rotation du coude

Après la suppression de la pince et du poignet, on obtient un modèle géométrique du robot extrêmement simple.

...

Ce modèle est représenté à la figure 5.20 et nous permettra de calculer l'angle de rotation de l'épaule et du coude.

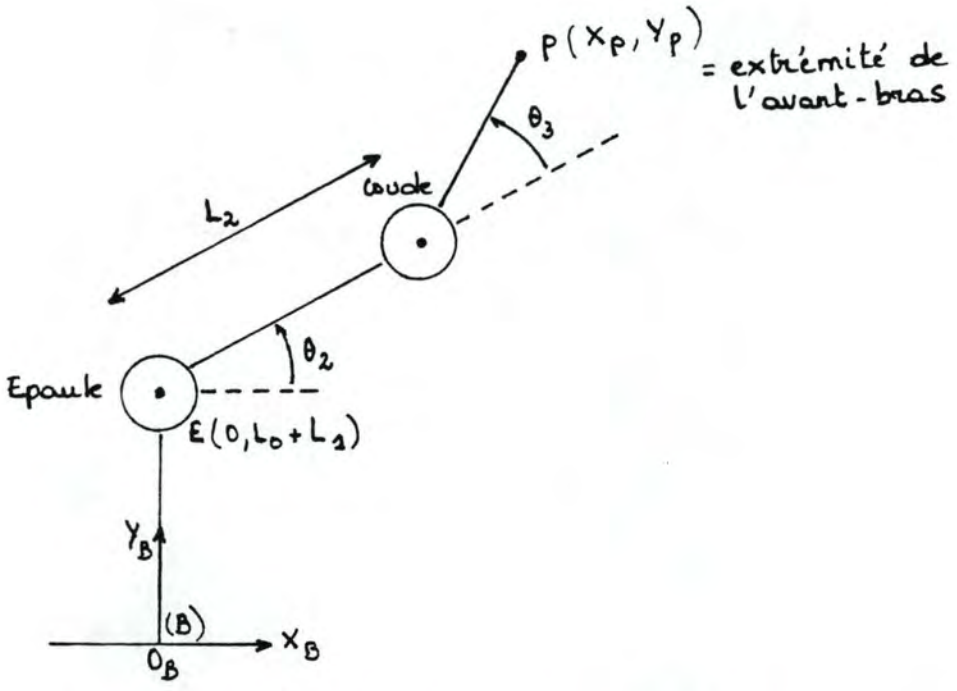



Fig. 5.20 - Modèle du robot RM 501 simplifié par la suppression de la pince et de l'axe du poignet de l'axe du corps.

L'axe de l'épaule, du coude et de l'extrémité de l'avant-bras forment les trois sommets d'un triangle quelconque. On peut trouver l'angle du coude en appliquant la relation de Pythagore à ce triangle.

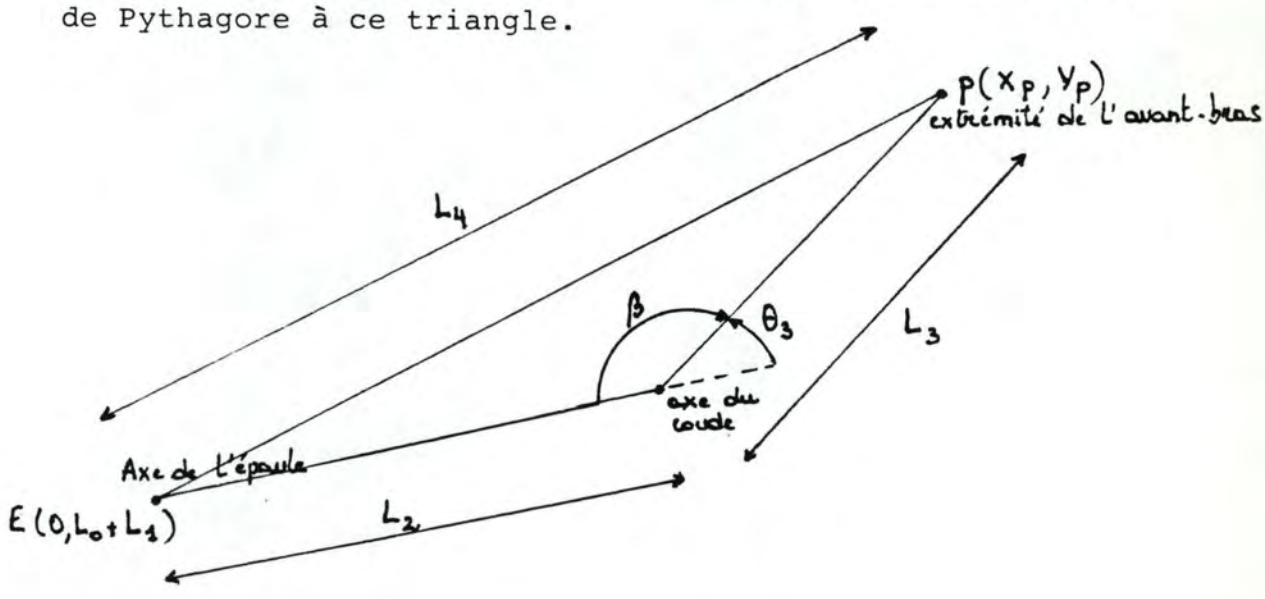



Fig. 5.21 - Calcul de l'angle du coude.

...

Par la relation de Pythagore, nous avons

$$L_4^2 = L_2^2 + L_3^2 - 2L_2L_3\cos\beta$$

$$= L_2^2 + L_3^2 - 2L_2L_3\cos(180 - \theta_3)$$

$$= L_2^2 + L_3^2 + 2L_2L_3\cos(\theta_3)$$

on obtient par mise en évidence de $\cos(\theta_2)$

$$\cos(\theta_3) = \frac{L_4^2 - L_2^2 - L_3^2}{2L_2L_3}$$

$$= \frac{X_p^2 + (Y_p - L_o + L_1)^2 - L_2^2 - L_3^2}{2L_2L_3}$$

et nous avons enfin

$$\theta_3 = \arccos\left[\frac{X_p^2 + (Y_p - L_o + L_1)^2 - L_2^2 - L_3^2}{2L_2L_3}\right]$$

Sous-problème 7 : Calcul de l'angle $\theta_2$ de rotation de l'épaule

La figure 5.21 nous permet encore de trouver l'angle de rotation de l'épaule, car nous pouvons raisonner sur le triangle défini par l'axe de l'épaule, l'axe du coude et l'extrémité de l'avant-bras.

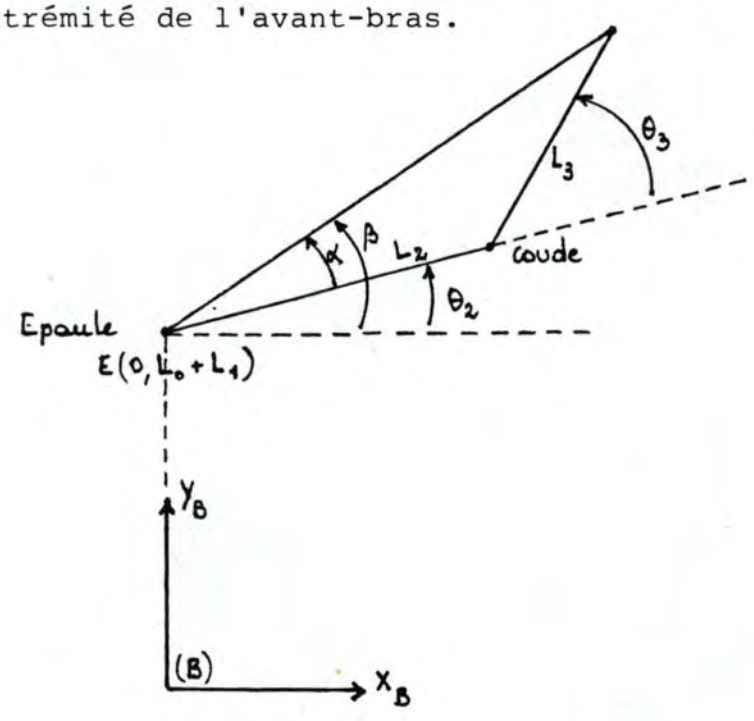



Fig. 5.22 - Calcul de l'angle de l'épaule.

...

L'angle $\theta_2$ est obtenu par la soustraction de l'angle à l'angle ß de la figure 5.11.

* calcul de l'angle ß

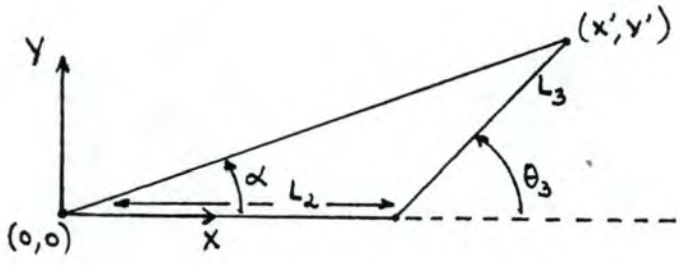
L'angle ß est aussi l'angle formé par la droite passant par les points e et p par rapport à l'axe $X_B$ du repère B.

On obtient alors la valeur de ß par

$$ß = \text{arctg}\left[\frac{Y_p - L_o - L_1}{X_p}\right]$$

* calcul de l'angle $\alpha$

L'angle $\alpha$ est l'angle formé par l'hypothénuse du triangle et son côté adjacent $L_2$. La méthode utilisée sera identique à celle utilisée pour le calcul.




On utilisera un triangle identique au précédent mais le côté caractérisé par la longueur $L_2$ se situe sur l'axe des X.

$$\alpha = \text{arctg}\left[\frac{Y'}{X'}\right]$$

avec

$$X' = L_2 + L_3\cos\theta_3$$

$$Y' = \sin\theta_3$$

on obtient

$$\alpha = \text{arctg}\left[\frac{\sin(\theta_3)}{L_2 + L_3\cos(\theta_3)}\right]$$

...

* calcul de l'angle $\Theta_2$

$$\Theta_2 = \beta - \alpha$$

donc

$$\Theta_2 = \text{arctg}\left[\frac{Y_b - L_o - L_1}{X_p}\right] - \text{arctg}\left[\frac{\sin(\Theta_3)}{L_2 + L_3\cos(\Theta_3)}\right]$$

Théoriquement, il y a deux solutions. La première correspond à une configuration où le coude est situé "vers le bas", la deuxième correspond à une configuration où le coude se situe "vers le haut". On ne retiendra que la deuxième, car la première est incompatible avec le mécanisme du robot.

Sous-problème 8 : Calcul de l'orientation du poignet

Si $\Theta_4$ représente l'inclinaison du poignet;
$\Theta_5$ représente la torsion du poignet

Connaissant $\Theta_2$ et $\Theta_3$ , le tangage et le roulis, on obtient pour l'orientation du poignet :

$$\Theta_4 = \text{Tangage} - \Theta_2 - \Theta_3$$

$$\Theta_5 = \text{Roulis}$$

5.1.3.2.3. Construction d'un module d'implémentation de la méthode "sur mesure"

Voir Annexe C.

...

## 5.2. Concepts inhérents à la programmation de niveau "XYZ"

La couche "XYZ" fait apparaître le concept de robot "XYZ". Le robot "XYZ" assure l'indépendance de sa programmation par rapport aux caractéristiques géométriques et mécaniques du robot articulaire. Le robot "XYZ" pourrait aussi s'appeler "pince" car la position et le déplacement du bout de la pince entrent en ligne de compte tandis que les positions des autres organes nous sont devenus complètement indifférents.

Il faut toutefois émettre certaines réserves. L'indépendance géométrique n'est pas totale. En effet, la pince ne peut se déplacer que dans le volume de l'espace de travail du robot articulaire.

Cette dépendance est exprimée implicitement par le transformateur de coordonnées dont nous avions parlé antérieurement. Ce dernier est chargé, notamment, de déterminer la configuration des organes du robot permettant de positionner l'extrémité de sa pince à une position exprimée à l'aide de coordonnées cartésiennes. Si la configuration obtenue est impossible pour le robot articulaire, cela signifie que la position cartésienne spécifiée n'est pas accessible par le bout de la pince.

La programmation du niveau "XYZ" permet d'ignorer la présence et la position du corps, de l'arrière-bras et de l'avant-bras du SMA. Cette simplification comporte certains risques car ces organes peuvent provoquer des collisions incontrôlées avec les objets disposés dans l'espace de travail.

On utilisera des positions intermédiaires pour remedier à ce problème. Ces positions intermédiaires, aussi appelées "points via", seront des positions cartésiennes par lesquelles la trajectoire de l'extrémité de la pince devra obligatoirement passer pour éviter les collisions. Les "points via" utilisés pour éviter la collision du corps, de l'arrière-bras et de l'avant-bras avec les objets disposés dans l'espace de travail sont caractéristiques de la géométrie et du nombre d'organes du robot articulaire, si on utilise un autre robot articulaire pour effectuer une même manipulation automatique, les "points via" de sa description programmée seront certainement différents.

...

La programmation du niveau "XYZ", malgré ces restrictions, est toutefois relativement portable car n'importe quel robot articulaire caractérisé par un nombre suffisant de degrés de liberté et par une géométrie relativement identique à celle du robot articulaire actuel, est susceptible d'exécuter la même tâche sans modification majeure de la programmation.

La couche "XYZ" remplit également un deuxième rôle : elle assure la communication du robot "XYZ" avec le robot "articulaire" de la couche inférieure par objets interposés (voir fig. 5. ).

La couche "XYZ" comporte un ensemble d'objets, chacuns caractérisés par une liste de compétences. Le robot "XYZ" communique avec le robot "articulaire" en envoyant des messages aux objets définis dans la couche.

L'activation des compétences des objets cibles des messages provoque l'envoi de messages au robot articulaire.

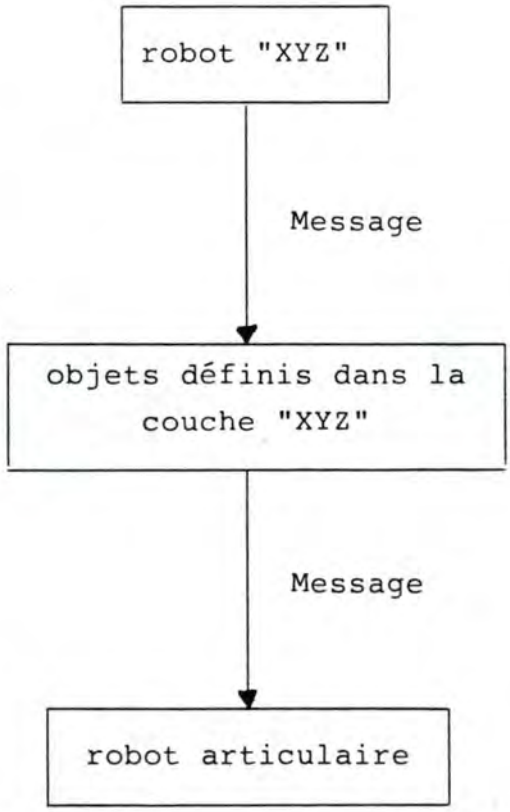



Fig. 5.23 - Communication du robot "XYZ" avec le robot articulaire.

...

### 5.2.1. Objets définis dans la couche "XYZ"

Les objets définis dans la couche "XYZ" sont représentés de la même manière qu'à la figure 4.2.

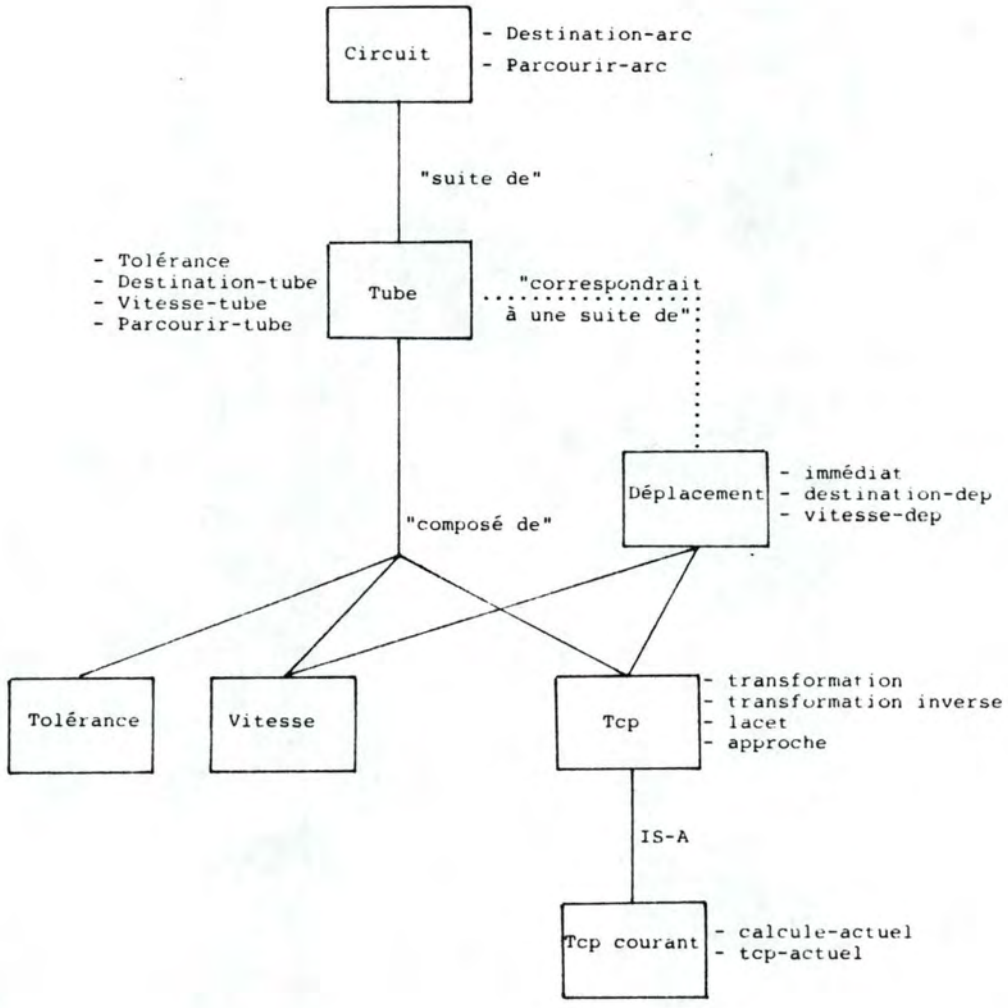



Fig. 5.24 - Structuration des objets définis dans la couche "XYZ"

...

5.2.2. Définition des objets de la couche "XYZ"

L'objet TCP : le concept de tcp est un concept central de la programmation de niveau "XYZ". En effet, le TCP (Tool Center Point) représente le repère associé à l'extrémité de la pince. Rappelons que ce repère définit la position et l'orientation de l'extrémité de la pince par rapport au repère de l'espace $R_o$ de travail. Les axes du repère de la pince sont orientés selon le schéma suivant :

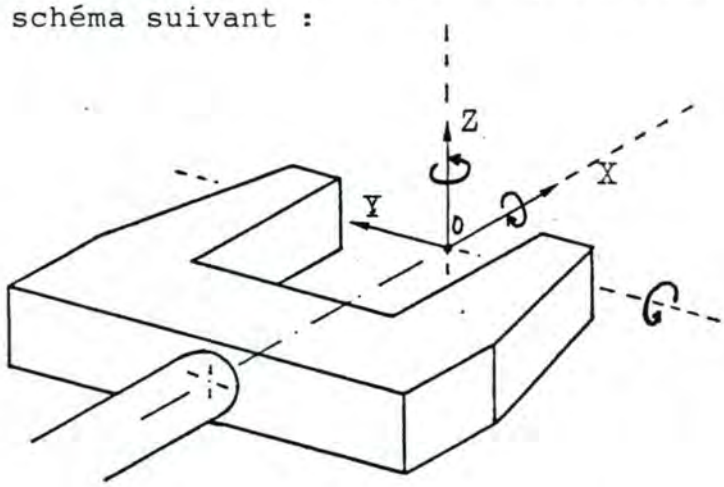



Fig. 5.25 - Orientation des axes du repère.

Le TCP sera constitué des coordonnées de l'origine O du repère de la pince par rapport au repère $R_o$ de l'espace de travail et de trois angles de rotation (tangage, roulis et lacet) qui caractérisent l'orientation de ce repère.

Les coordonnées du point O seront exprimées par des entiers en centièmes de mm et les angles de rotation par des entiers en centièmes de degrés.

La programmation du robot consistera essentiellement à faire coïncider le repère de l'outil avec d'autres repères liés à la tâche à exécuter. Dans le cas de la manipulation d'objets, l'utilisateur devra donc définir pour chaque objet un ou plusieurs repères de saisie en tenant compte des deimensions géométriques et du sens d'approche de la pince pour que son repère vienne coïncider avec le repère de saisie (fig. 5. ).

Le concept de TCP peut être élargi. On peut l'utiliser pour représenter les repères de saisie des objets et les repères définissant leur position et leur orientation.

...

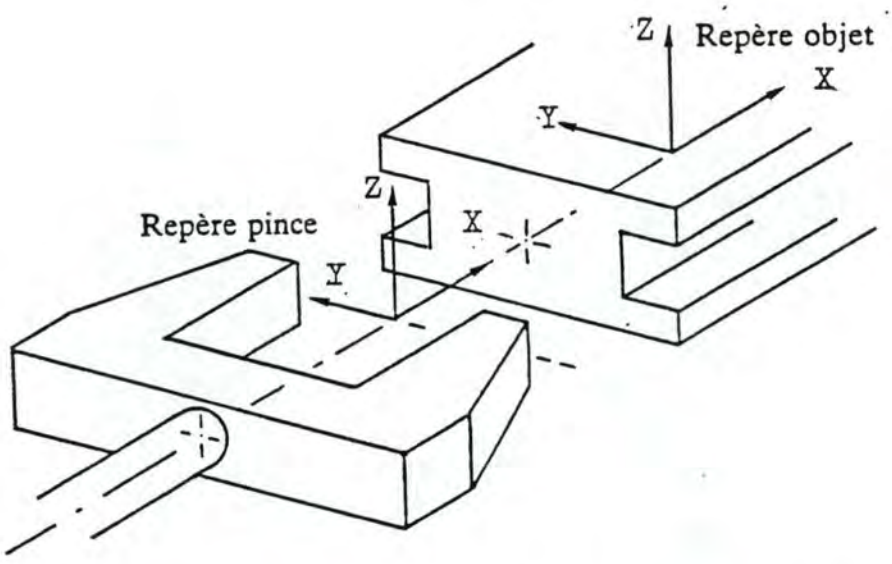



Fig. 5.26 - Illustration de la saisie d'un objet.

Le TCP courant : est un tcp qui représente la position et l'orientation courantes de l'extrémité de la pince par rapport au repère $R_o$ de l'espace de travail du robot.

La vitesse : identique au concept de vitesse défini dans la couche articulaire.

Le déplacement : représente un mouvement du tcp de la pince vers un tcp cible donné par un objet de classe tcp et avec une vitesse donnée par un objet de classe vitesse.
Ce concept correspond au concept de mouvement de la couche articulaire.

La tolérance : détermine l'écart maximal que peut avoir la trajectoire réelle de la pince par rapport à la trajectoire théorique. Elle s'exprime en centièmes de millimètres par un nombre entier.

Le tube : est une enveloppe géométrique couvrant les trajectoires possibles à suivre par le tcp de la pince du robot pour atteindre un tcp cible. L'axe de symétrie de l'enveloppe cylindrique est une droite joignant le tcp courant de la pince avec le tcp cible. Cette droite représente également la trajectoire théorique(rectiligne) de déplacement du tcp de la pince. Le tcp cible

...

du tube est donné par un objet de classe tcp. Le diamètre du tube est donné par un objet de classe tolérance. Cette tolérance représente donc l'écart maximal de la trajectoire réelle du tcp de la pince par rapport à l'axe de symétrie du tube. Enfin, le parcours de la trajectoire réelle de la pince se fait à une vitesse donnée par un objet de classe vitesse.

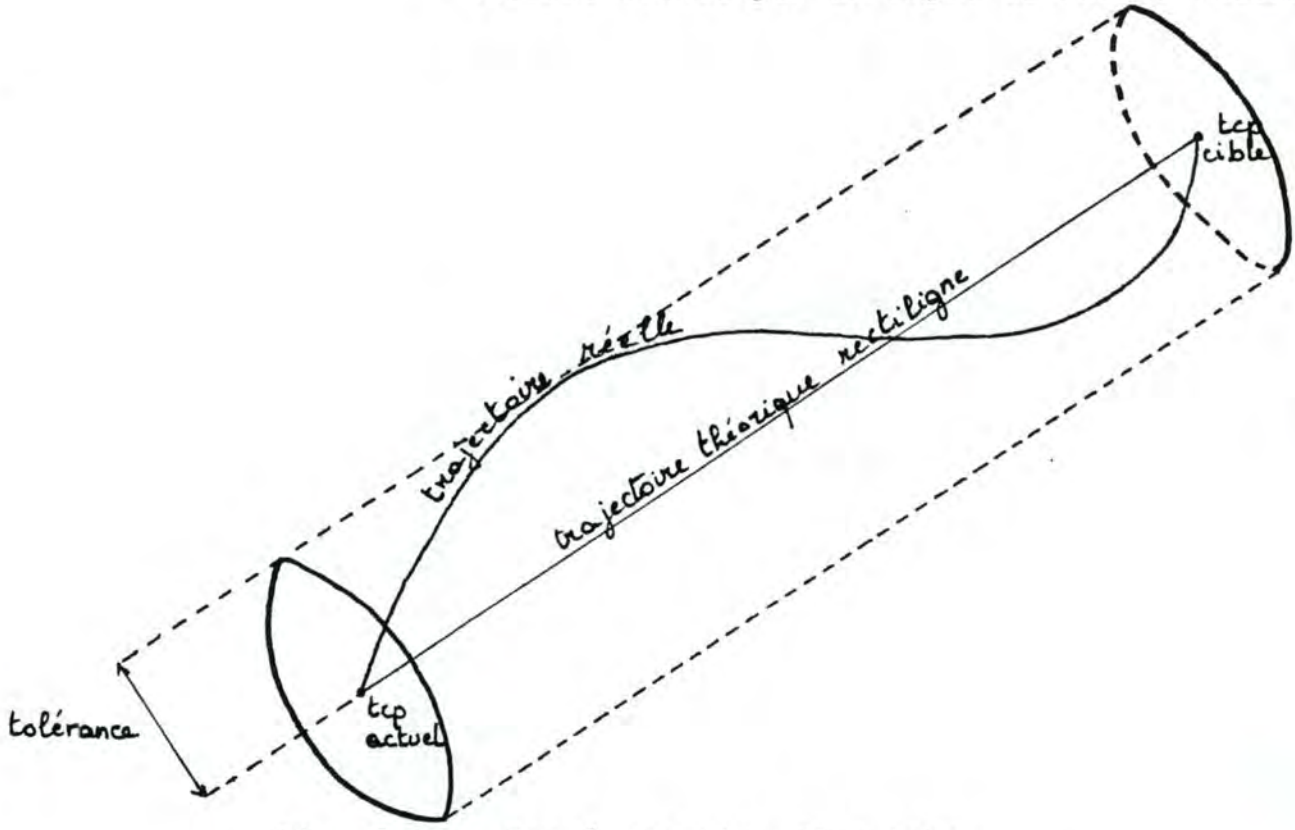



Fig. 5.27 - Représentation d'un tube.

Discussion à propos du choix de la géométrie d'une trajectoire

Nous avons déjà défini deux concepts représentant des trajectoires :

- Le concept de déplacement : représente une trajectoire libre. Une trajectoire est libre lorsqu'elle n'est soumise à aucune loi géométrique.
  Le déplacement représente également le chemin le plus rapide mais pas nécessairement le plus court pour déplacer le tcp de la pince de sa position initiale vers une position cible. Cette trajectoire est donc très souple mais peu précise car elle n'est pas contrôlable.

...

- Le concept de tube : représente une trajectoire rectiligne ou quasiment rectiligne.

Nous avons choisi la trajectoire rectiligne pour plusieurs raisons :

(1) les lignes droites décrivent une trajectoire prévisible et facile à visualiser;
(2) elles donnent le chemin le plus court (mais pas nécessairement le plus rapide) entre une source et une destination;
(3) les trajectoires rectilignes sont utilisées dans bon nombre d'applications; par exemple, l'insertion d'une cheville dans un trou necessite une trajectoire rectiligne;
(4) selon Richard Paul (R.P. Paul - 1981), les trajectoires rectilignes permettent de minimiser les forces dues à l'inertie de la pince et de l'objet transporté;
(5) les trajectoires rectilignes permettent de simplifier le module chargé de la planification de la trajectoire.

La tolérance ajoute une souplesse supplémentaire à la trajectoire rectiligne. Si la trajectoire de la pince est plus libre, celle-ci aura tendance à suivre une trajectoire dictée par la géométrie du bras qui la porte. Cette liberté de mouvement est facilement contrôlable car elle est limitée par l'enveloppe que constitue le tube.

Dans notre approche, on concilie donc les avantages de la trajectoire libre et de la trajectoire rectiligne.

Liens entre le tube et le déplacement

- Le tube contient toutes les informations nécessaires pour calculer une suite de points intermédiaires se situant sur l'axe de symétrie de l'enveloppe cylindrique qu'il représente. Ces points intermédiaires sont des points de passage obligatoires de la trajectoire réelle du tcp de la pince, de manière à ce que celle-ci se situe toujours à l'intérieur de l'enveloppe cylindrique que représente le tube. Ces points intermédiaires seront tous equidistants et en nombre inversément proportionnel à la tolérance du tube.

...

En définitive, le parcourspar le tcp de la pince d'une des trajectoires inscrites dans l'enveloppe du tube se traduira par le passage successif du tcp de la pince par toutes les positons intermédiaires avec une vitesse qui caractérise celle du tube. Or, le passage du tcp de la pince d'une position à une autre avec une vitesse donnée correspond au concept de déplacement. La trajectoire contenue dans le tube pourra donc être vue comme une suite de déplacements sans interruption du mouvement de la pince entre chaque déplacement.

- Un tube caractérisé par une tolérance infinie est un déplacement.

Le circuit : représente une trajectoire composée d'une suite de tubes. Le passage d'un tube à l'autre ne doit pas nécessairement provoquer un arrêt du mouvement de la pince. Les tcp délimitant les tubes sont des points de passage obligatoire de la trajectoire.

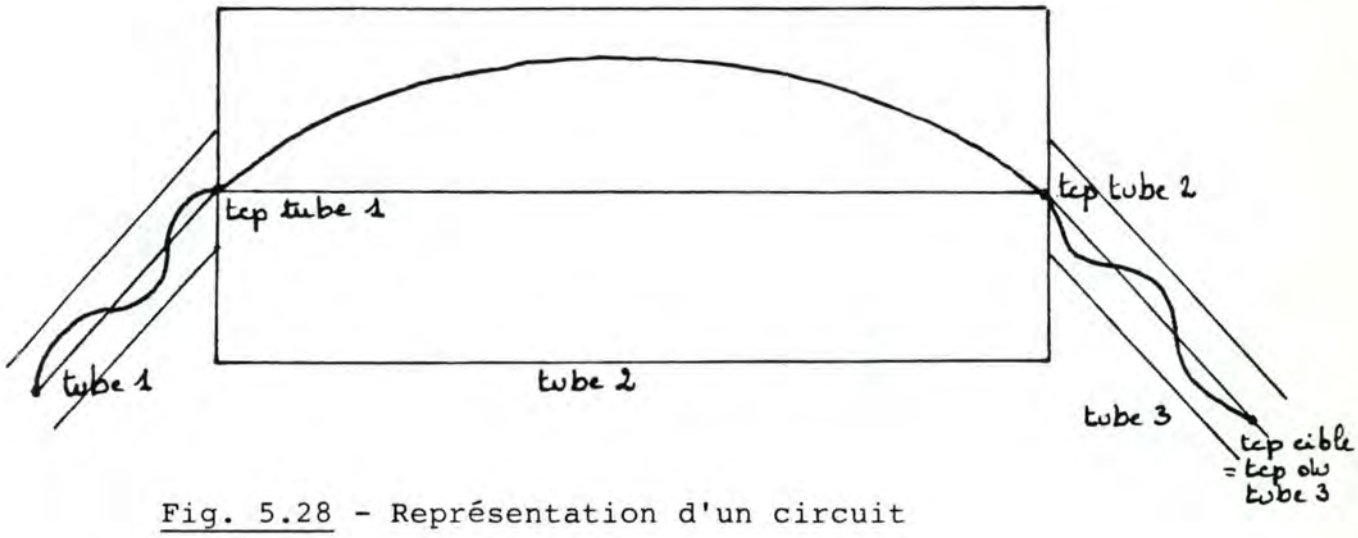



Fig. 5.28 - Représentation d'un circuit

La force : concept identique au concept de force défini dans la couche articulaire.

Le temps : concept identique au concept de temps défini dans la couche articulaire.

La position articulaire : il s'agit du concept de position défini dans la couche articulaire. Le mot "articulaire" a été ajouté par souci d'éviter toute ambiguïté.

...

### 5.2.3. Correspondance des concepts du niveau articulaire et du niveau "XYZ"

Nous pouvons faire correspondre certains concepts définis dans la couche articulaire avec des concepts de la couche "XYZ".

Position ⟷ tcp : La position représente une configuration des organes du SMA. A une configuration donnée correspond une position et une orientation bien précise de la pince dans l'espace de travail. Le tcp représente la position et l'orientation d'un repère solidaire de l'extrémité de la pince. Ce repère identifie aussi la position et l'orientation de la pince dans l'espace de travail. Le concept de position correspond bien au concept de tcp.

Mouvemement ⟷ déplacement : Le mouvement était défini dans la couche articulaire comme le passage de la position actuelle du SMA vers une position cible avec une vitesse donnée. Le déplacement est défini dans les mêmes termes mais on utilise un tcp au lieu d'une position. Comme nous avons établi la correspondance entre la position et le tcp, il est donc facile d'établir la correspondance entre le mouvement et le déplacement.

Segment ⟷ tube : Le segment est une suite de mouvement mais malheureusement, le tube n'est pas une suite de déplacements! Dans une remarque précédente, j'ai souligné le fait que la trajectoire enveloppée dans le tube pouvait être vue comme une suite de mouvements, tous caractérisés par la même vitesse. Il y a donc une correspondance entre les concepts de tube et de segment. Seulement, cette correspondance est moins évidente.

Trajectoire ⟷ circuit : La correspondance est très nette : une trajectoire est une suite de segments et un circuit, une suite de tubes.

### 5.2.4. Compétences des objets définis dans la couche "XYZ"

#### 5.2.4.1. Compétences du tcp

TCP

Requiert : six paramètres
Objet créé : tcp
Effet : crée et renvoie une instance de tcp en utilisant les six paramètres requis. Ces paramètres re-

...

présentent respectivement
- les coordonnées X,Y,Z de l'origine du repère représenté par le tcp créé. Ces coordonnées sont exprimées en centièmes de millimètres par rapport au repère $R_o$ de l'espace de travail;
- le tangage, le roulis et le lacet du repère représenté par le tcp créé. Ces trois angles sont exprimés en centièmes de degrés.

TRANSFORMATION

Objet recquis : position articulaire
Objet créé : tcp
Effet : calcule et renvoie un tcp. Ce tcp représente un repère qui exprime la position et l'orientation de l'extrémité de l'outil, connaissant la configuration des organes qui le portent. On utilisera pour ce calcul le modèle de la cinématique directe du robot RM 501 présenté au paragraphe 5.1.3.1.7.

TRANSFORMATION INVERSE

Objet créé : position articulaire
Effet : renvoie une position articulaire correspondant à une configuration des organes du SMA. Cette configuration permet le positionnement et l'orientation de l'extrémité de l'outil correspondant à l'objet de classe tcp dont on a activé la compétence "transformation inverse". On utilisera pour le calcul de la position articulaire, le modèle de la cinématique inverse du robot RM 501 présenté au paragraphe 5.1.3.2.2.

LACET

Objet créé : entier
Effet : calcule et renvoie le lacet du repère représenté par l'objet de classe tcp dont on a activé la compétence "lacet". Ce lacet est exprimé en centièmes de degrés par un nombre entier.
Intérêt : cette compétence nous est utile car le robot RM 501 est démuni d'axe de lacet. Par conséquent, le lacet du repère associé à l'extrémité

...

de la pince sera toujours fonction de la position de son origine par rapport au repère $R_o$ de l'espace de travail. Nous considérerons donc cette compétence comme une option spécifique aux robots démunis d'axe de lacet.

APPROCHE

Requiert : deux paramètres

Objet créé : tcp

Effet : calcule et renvoie un objet de classe tcp qui représente un repère d'axes $X_i Y_i Z_i$ respectivement parallèles aux axes $X_j Y_j Z_j$ du repère représenté par l'objet de classe tcp dont on a activé la compétence "approche". Les axes $X_i$ et $X_j$ des deux repères doivent toujours se situer dans un même plan vertical. Les deux paramètres requis sont deux nombres entiers qui représentent respectivement

- la distance séparant l'origine des deux repères représentés par les deux tcp. Cette distance est exprimée en centièmes de millimètres par un nombre entier.
- la pente par rapport au repère $R_o$ de l'espace de travail de la droite joignant l'origine des repères des deux tcp.

Exemple :

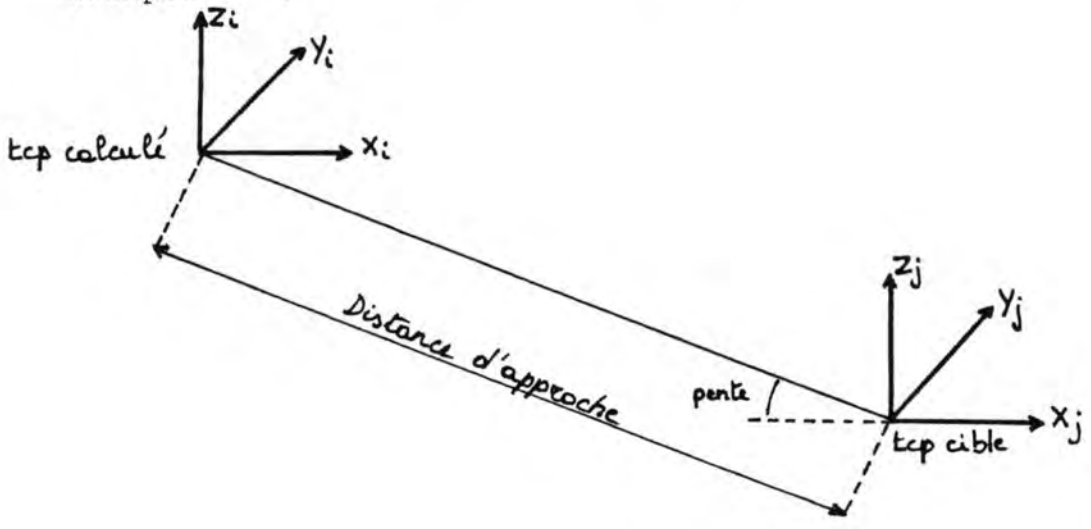



Fig. 5.29 - Illustration de l'approche

...

Intérêt : il est très souvent utile d'approcher le tcp de la pince vers une position finale. Cette approche constituera, par exemple, la phase finale d'une trajectoire de déplacement du tcp de la pince permettant la saisie d'un objet. Cette phase finale est définie à l'aide d'un point intermédiaire (point d'approche) qui ne se situe pas nécessairement sur la trajectoire directe en provenance du point de départ de l'extrémité de la pince.
L'approche d'un objet est illustrée à la fig. 5.30. et dans les applications traitées au chapitre 6.

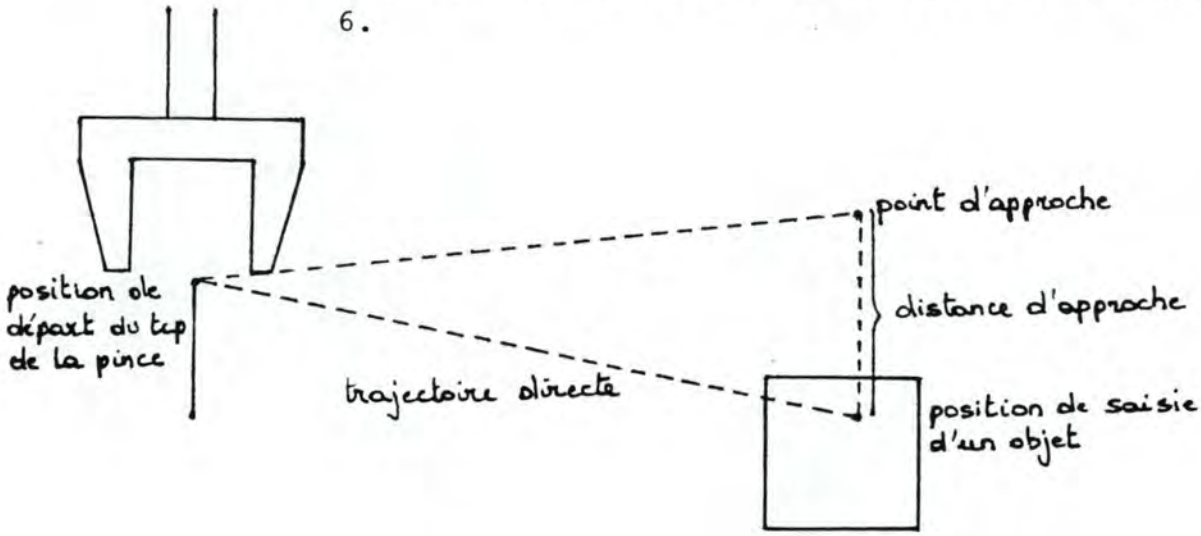



Fig. 5.30 - Illustration d'une trajectoire d'approche d'un objet par la pince.

Dans les applications traitées au chapitre 6, nous utiliserons un circuit constitué de deux tubes pour définir une telle trajectoire d'approche. Le premier tube sera caractérisé par une tolérance très grande et définira une trajectoire de déplacement du tcp de la pince sur le point d'approche. Le deuxième tube, caractérisé par une faible tolérance, assurera la définition d'une trajectoire fort rectiligne du déplacement de tcp de la pince sur la position de saisie d'un objet à déplacer.

...

CALCULE-ACTUEL

Objet créé : tcp

Effet : renvoie un objet de classe tcp qui représente la position et l'orientation du repère associé à l'extrémité de la pince. Cette position et cette orientation sont exprimées par rapport au repère $R_o$ de l'espace de travail.

Méthode : - active la compétence "pos-actuelle" de la classe "position" définie dans la couche articulaire afin de connaître la position articulaire actuelle du robot.
- active la compétence "transformation" de la classe tcp et donne comme argument la position articulaire.
- renvoie le tcp obtenu.
- met à jour le tcp actuel.

Intérêt : On utilise cette compétence lorsque le tcp identifiant la position et l'orientation actuelles du repère associé à la pince est inconnu du système. Cette situation se présente lors de l'initialisation du robot ou après la modification directe de la configuration des organes du SMA par un programme du niveau "XYZ".

TCP-ACTUEL

Objet créé : tcp

Effet : renvoie un objet de classe tcp qui représente la position et l'orientation du repère associé à l'extrémité de la pince.

Intérêt : on utilisera cette compétence lorsque l'usage de la compétence "calcule-actuel" n'est pas nécessaire pour connaître le tcp actuel de la pince.

5.2.4.2. Compétences du déplacement

IMMEDIAT

Effet : provoque le déplacement immédiat du tcp de la pince vers un tcp cible avec une vitesse donnée. Le tcp cible et la vitesse sont donnés par un objet de classe déplacement dont on a activé la

...

compétence "immédiat".

DESTINATION-DEP

Objet créé : tcp
Effet : renvoie le tcp de l'objet de classe "déplacement" dont on a activé la compétence "destination-dep".

VITESSE-DEP

Objet créé : vitesse
Effet : renvoie la vitesse de l'objet de classe déplacement dont on a activé la compétence "vitesse-dep".

5.2.4.3. Compétences du tube

TOLERANCE

Objet créé : tolérance
Effet : renvoie la tolérance du tube.

DESTINATION-TUBE

Objet créé : tcp
Effet : renvoie le tcp du tube.

VITESSE-TUBE

Objet créé : vitesse
Effet : renvoie le tcp du tube.

PARCOURIR-TUBE

Effet : provoque le déplacement du tcp de la pince sur un tcp cible et avec une vitesse donnés par le tube; selon une trajectoire inscrite dans l'enveloppe cylindrique que représente le tube.

Méthode : - calcule une série de points intermédiaires qui constitueront des points de passage obligatoires de la trajectoire du centre de la pince, de manière à ce que la déviation de cette trajectoire par rapport à la trajectoire théorique n'excède pas la tolérance autorisée. Pour le robot RM 501, on estime que le déplacement de la pince vers un tcp distant de 5 cm donne lieu à une déviation d'1 mm.
- détermine la position articulaire de tous les

...

tcp de la trajectoire en activant leur compétence "transformation-inverse".

- crée un objet trajectoire à partir des positions articulaires calculées. La vitesse de tous les mouvements de la trajectoire est identique à celle du tube.
- active la compétence "parcourir" du robot articulaire et donne comme argument la trajectoire créée.
- met à jour le tcp actuel.

Exemple : soit un tube caractérisé par une tolérance de 2 mm. La distance entre le tcp actuel et le tcp du tube est de 25 cm. On calculera deux points intermédiaires distants de 10 cm.

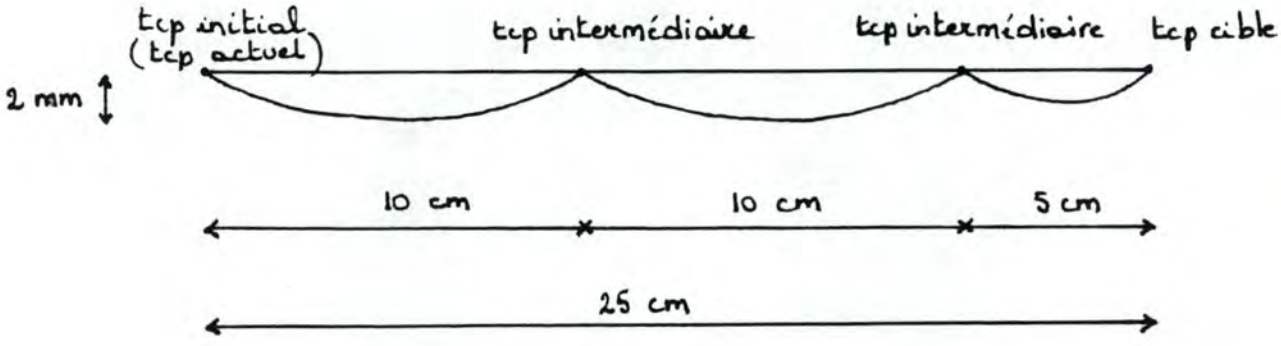



Fig. 5.31 - Calcul des positions intermédiaires d'une trajectoire délimitée par un tube.

5.2.4.4. Compétences du circuit

DESTINATION-CIRC

Objet créé : tcp

Effet : donne le tcp du dernier tube du circuit.

PARCOURIR-CIRC

Effet : provoque le parcours d'une trajectoire par le tcp. Cette trajectoire est définie par la succession des tubes du circuit.

...

### 5.2.5. Objets définis dans l'interface "XYZ"

Etant donné que nous nous sommes déjà étendus suffisamment longtemps sur la définition et les compétences de ces objets, nous nous limiterons à dresser la liste de ces objets :

- Tcp;
- Déplacement;
- Circuit;
- Tube.

### 5.2.6. Implémentation de la couche "XYZ"

Voir Annexe D

# 6. QUELQUES PROGRAMMES D'APPLICATION DU NIVEAU "XYZ"

## 6.1 DESCRIPTION D'UN ATELIER FLEXIBLE D'ASSEMBLAGE DE PLAQUES

L'atelier automatique a pour objectif d'assembler un ensemble de plaques de verre. Un assemblage est constitué d'une plaque munie d'un joint d'étanchéité et d'une plaque ordinaire unies à l'aide de trois pinces à papier. Le montage terminé, on obtient un récipient plat destiné à l'observation de gels acryliques.

Cet atelier flexible sera utilisé par un laboratoire d'analyse d'un service médical de l'hôpital GASTHUISBERG de la K.U.L. à Louvain.

Cette application fonctionne à titre expérimental car les stations de distribution n'ont pas encore été construites. Actuellement, on utilise des montages en carton pour représenter ces stations. La station d'assemblage est posée sur un plateau tournant actionné par un moteur électrique. Ce plateau est réalisé à l'aide d'un jeu d'assemblage du type "mécano".

### 6.1.1. Description des composants du montage

#### 6.1.1.1. une plaque avec joint d'étanchéité

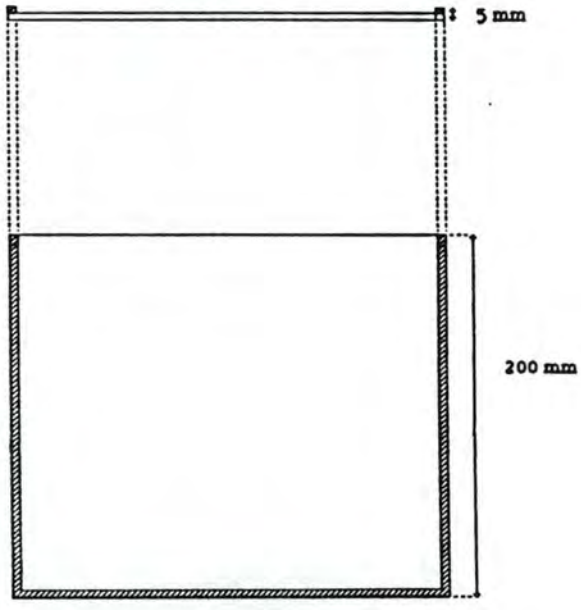



Fig. 6.1.

Dimension d'une plaque avec joint

Ces plaques sont fournies avec un joint d'étanchéité fixé à la vitre.

6.1.1.2. une plaque sans joint d'étanchéité

Cette deuxième plaque a exactement les mêmes dimensions que la première seulement, elle n'est pas munie d'un joint d'étanchéité.

6.1.1.3. trois pinces à papier

Les pinces à papier sont des pinces ordinaires que l'on peut trouver dans le commerce. Elles sont constituées d'un ressort et de deux leviers permettant son ouverture.

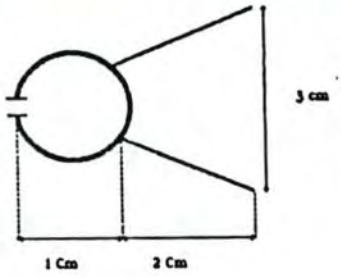



Fig.6.2 Dimensions d'une pince

Les pinces sont placées à mi-distance de chaque côté étanche de l'assemblage des deux plaques décrites ci-dessus.

6.1.1.4. résultat du montage

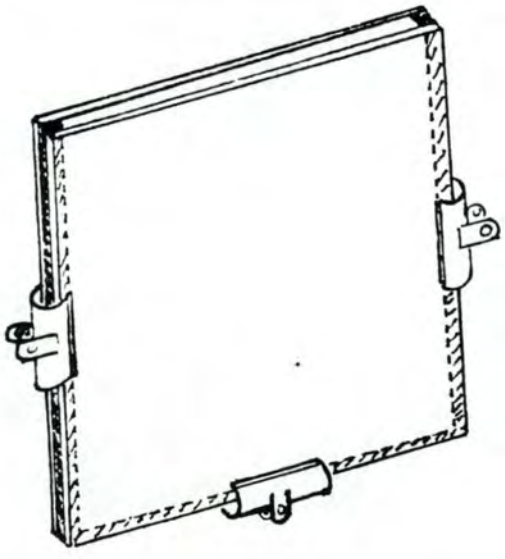


Fig.6.3. Résultat du montage

On obtient un petit aquarium plat. La partie supérieure reste ouverte afin de pouvoir y verser les gels à observer.

...

6.1.2. Description des stations de l'atelier flexible

On désigne par stations, les sites appartenant à l'espace de travail du robot dans lesquels sont accomplies des actions spécifiques.

L'atelier est constitué de cinq stations.

6.1.2.1. Les stations de distribution

Pour exécuter une tâche répétitive automatiquement à l'aide d'un robot aveugle et insensible, il est impératif que les objets manipulés soient à une position connue du système avant l'exécution d'un cycle de travail. Pour que cette contrainte soit satisfaite, on utilise généralement un distributeur d'objets. Ces distributeurs sont composés de deux parties :

- un magasin d'objets contenant une réserve d'objets. Cette réserve permet une certaine autonomie de l'atelier automatique;
- un mécanisme de présentation : ce mécanisme puise un objet du magasin du distributeur et le présente à la pince du robot toujours de la même manière. Le fait de présenter les objets dans la même position grâce à l'utilisation de distributeurs facilite l'aspect répétitif de la tâche.

6.1.2.1.1. la station de distribution de plaques sans joint

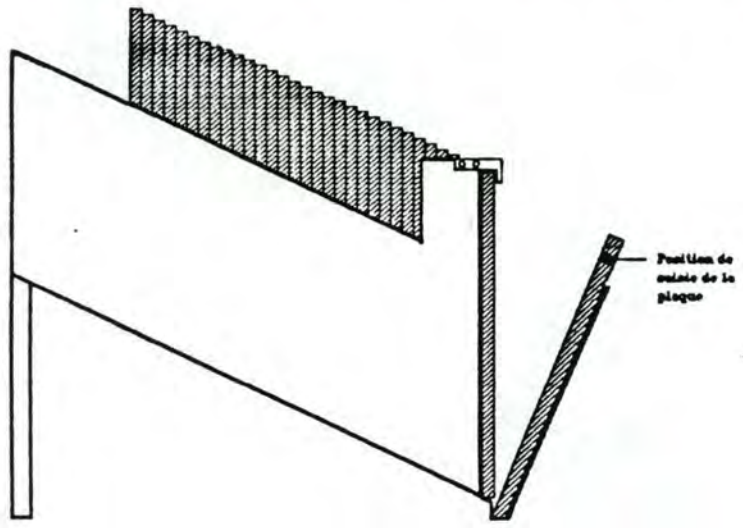


Fig. 6.4. Station de distribution de plaques de verre

...

6.1.2.1.2. la_station_de_distribution_de_plaques_avec_joint

Cette station est identique à la précédente, seul le mécanisme de présentation est un peu plus large, à cause de la présence du joint d'étanchéité. Le côté sans joint est présenté à la pince de manière à ne pas déteriorer les joints.

6.1.2.1.3. la_station_de_distribution_des_pinces

Les pinces sont emplilées dans un chargeur de pinces. Seule la pince située au sommet de la pile est accessible. Lorsque la pince située au sommet de la pile est enlevée, la pince qui lui succède prend sa place.

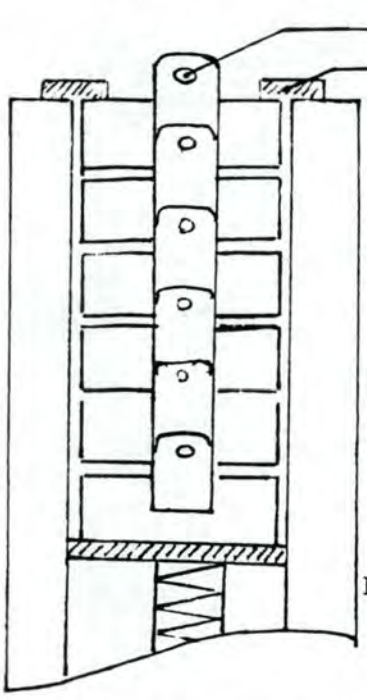



Fig. 6.5. Station de distribution des pinces

6.1.2.2. La station de dépôt

Lorsqu'un aquarium a été assemblé, il faut l'entreposer à un endroit fixe. On propose donc la création d'une station de dépôt des aquariums conçue de manière à ce que le site de dépôt du résultat du montage soit toujours identique pour chaque cycle d'exécution du travail.

(voir Fig. n° 6.6 - p. 6.5)

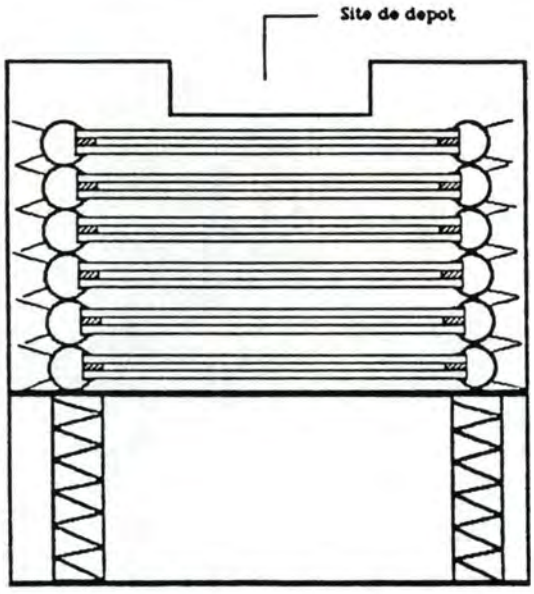



Fig. 6.6.

Station de depot des aquariums

Les aquariums sont mis en pile dans une boîte. Le fond de la boîte descend au fur et à mesure que le poids de la pile augmente. De cette manière, le site de dépôt reste inchangé pour chaque cycle d'exécution du travail.

6.1.2.3. La station de montage

La station de montage est un cadre posé sur un plateau tournant. Le plateau tournant sert à remplacer le degré de liberté manquant du poignet du robot.

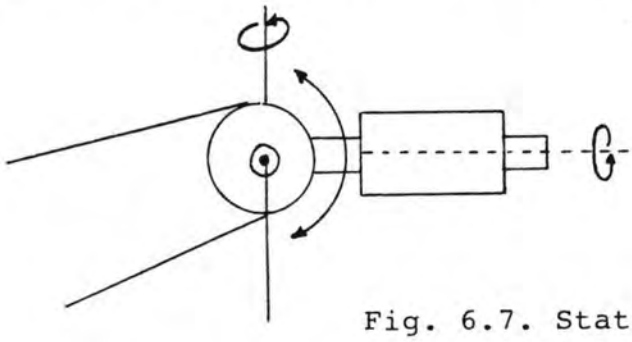
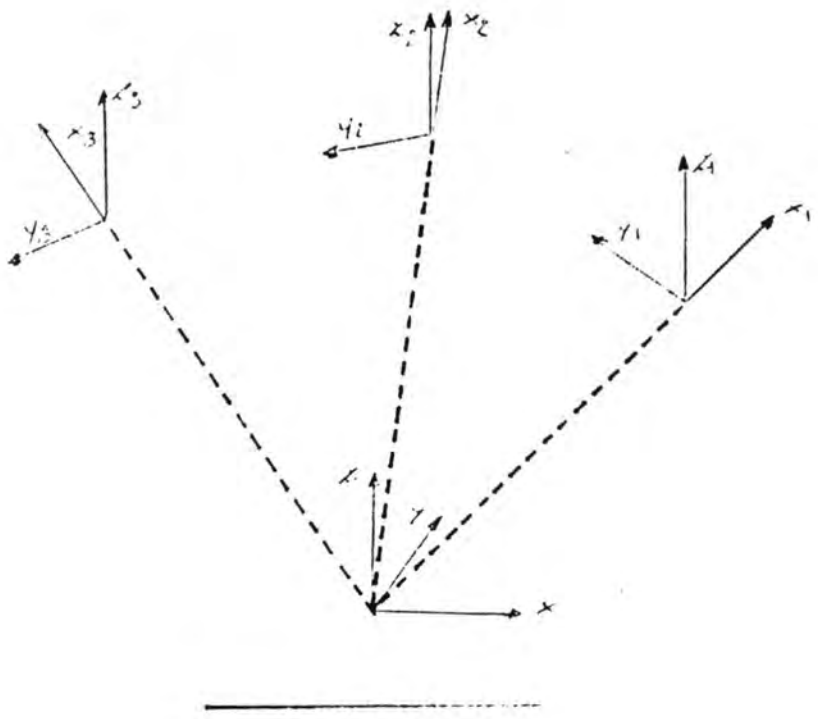
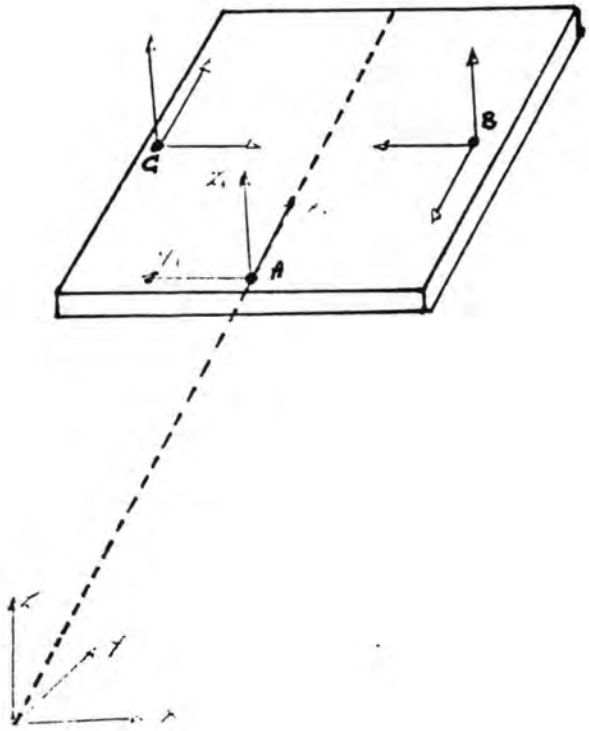


Fig. 6.7. Station de montage

A cause de l'absence de l'axe de lacet, les objets à manipuler doivent être orientés de manière à ce que leur angle de lacet soit toujours identique à l'angle de rotation du corps du robot RM 501.

...






Pour assembler un aquarium, on doit placer une pince en A,B et C. Seul le point A est accessible par la pince. Pour atteindre les points B et C, il faut les placer à la place du point A en pivotant l'ensemble du montage sur lui-même.

Fig. 6.8 - Contraintes imposées par l'absence d'axe de lacet

La station de montage doit donc être montée sur un plateau tournant de manière à ce que les points A, B et C de fixation des pinces soient accessibles tour à tour par la pince du robot.

...

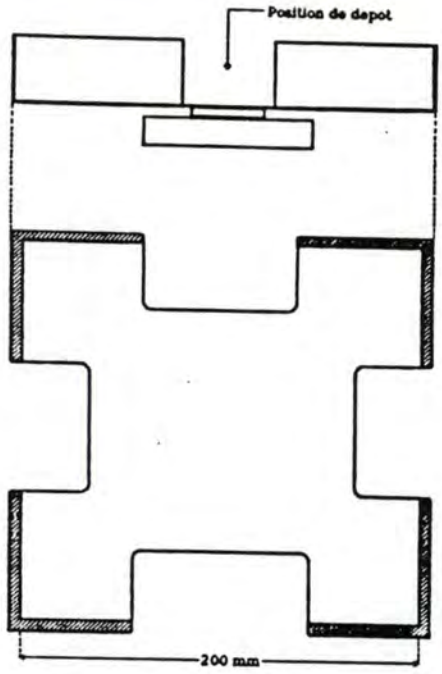



Fig. 6.9 - Station de montage des 'aquariums'

Les dimensions intérieures du cadre de montage doivent coïncider avec les dimensions des plaques de manière à permettre leur parfaite juxtaposition.

6.1.3. Disposition des stations et définitions des sites

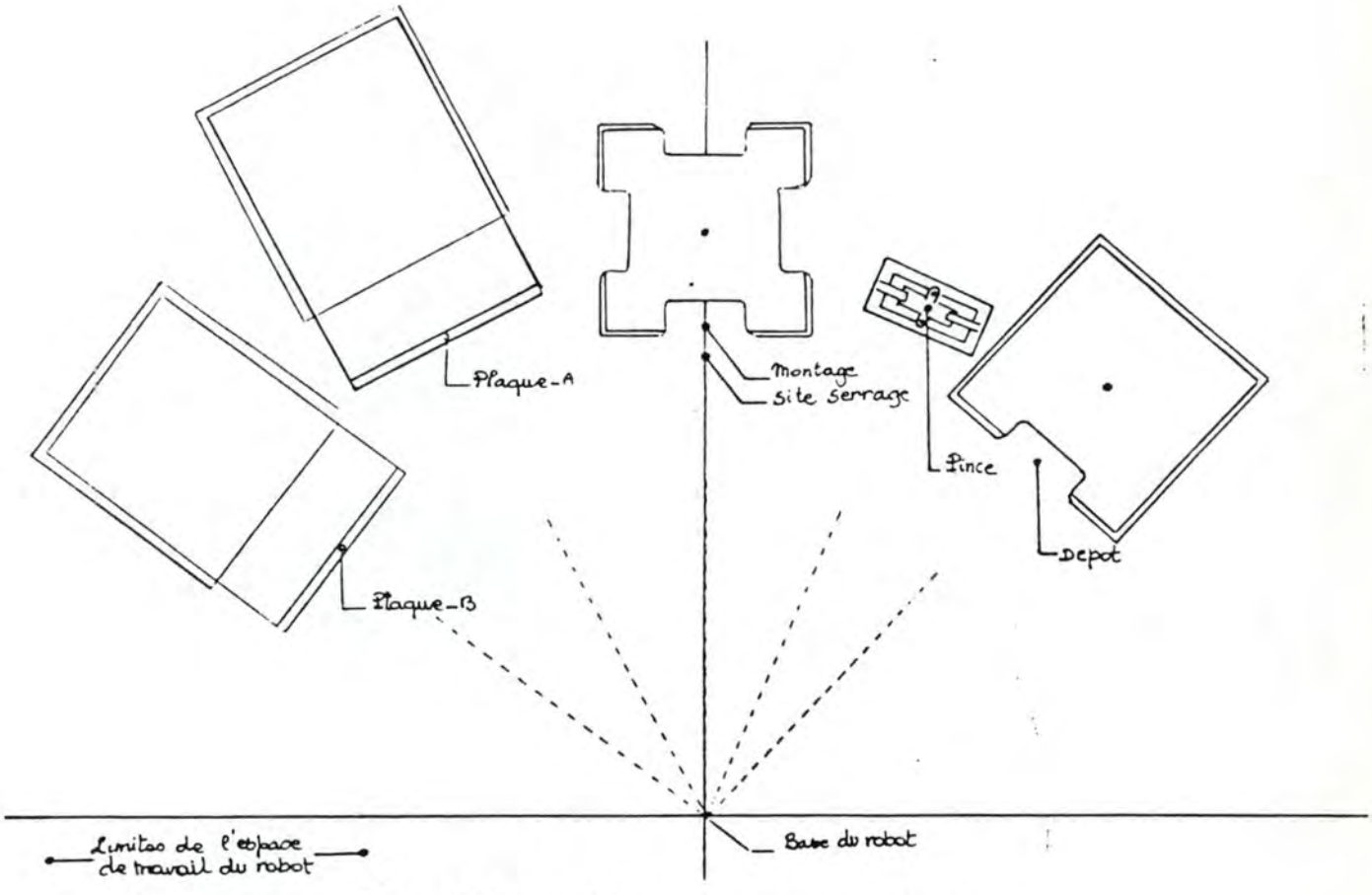


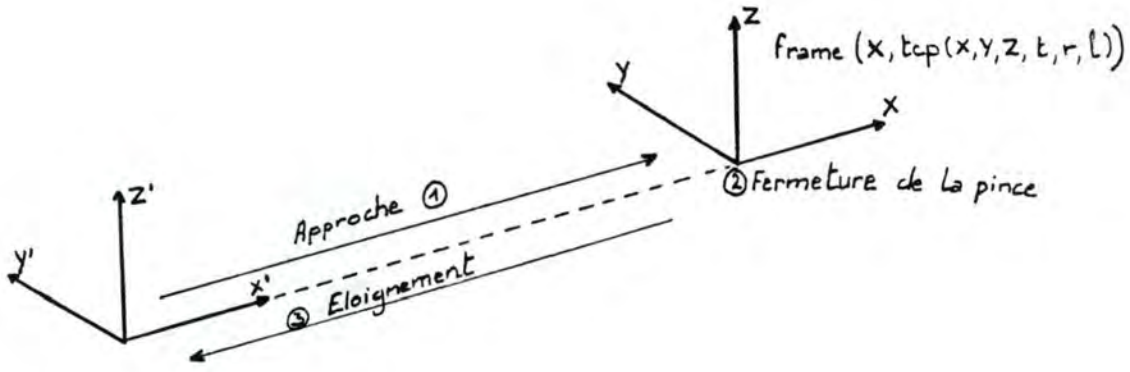

Fig. 6.10 - Disposition des sites

...

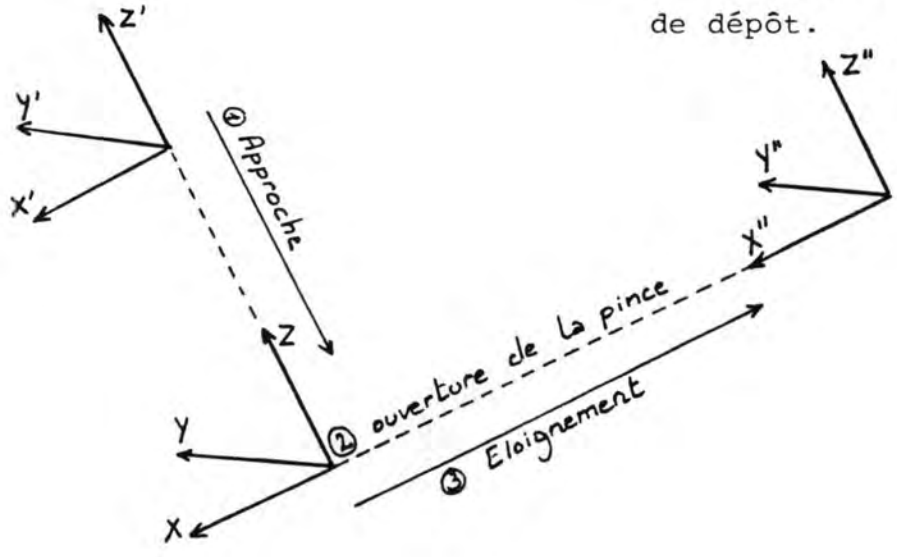
6.1.4. Définition des mouvements

Ces mouvements sont définis à l'aide de primitives similaires à celles que nous rencontrerons au niveau "objet". Il s'agit pourtant bien d'une programmation du niveau XYZ. Le programme contient une base de données élémentaires des objets. On y reprend seulement le nom des objets et leur position. C'est au programme d'application d'assurer la maintenance de cette base de données. Ce programme d'application nous permet donc de comprendre la nécessité d'un niveau "objet".

6.1.4.1. Saisir (X) utilisé pour saisir une plaque ou saisir une pince

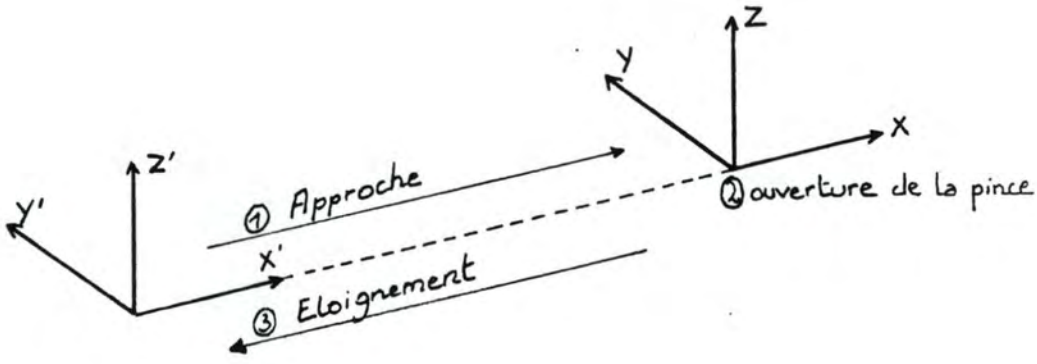



6.1.4.2. Déposer dans (X) utilisé pour déposer une plaque ou un montage dans la station de montage ou dans la station de dépôt.

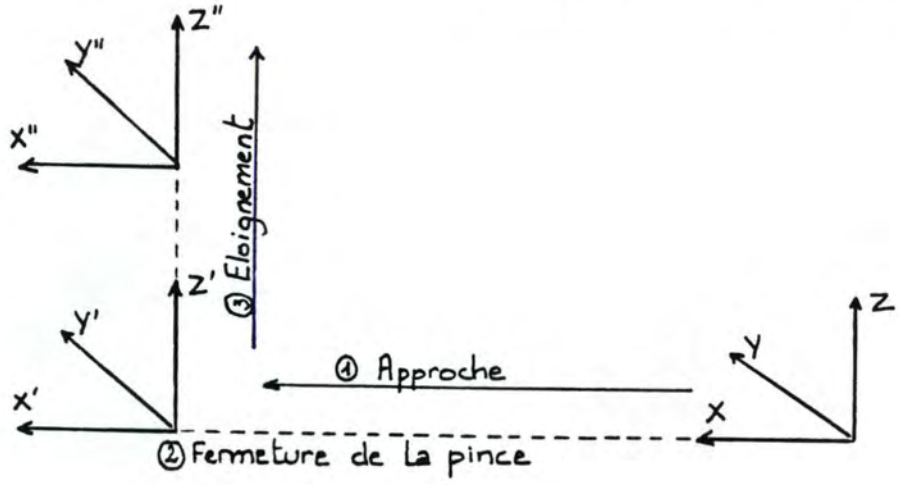



...

6.1.4.3. Fixer (X) utilisé pour fixer une pince sur un montage




6.1.4.4. Sortir de (X) utilisé pour sortir le montage de la station de montage




...

```
/*
Definition des coordonnees des sites
------------------------------------
*/
site(plaque_a,tcp(47750,0,16000,4000,9000,_)).
site(plaque_b,tcp(47750,0,16000,4000,9000,_)).
site(montage ,tcp(0,54300,13600,400,9000,_)).
site(depot   ,tcp(~54300,0,25600,0,9000,_)).
site(pince   ,tcp(32440,0,5300,9000,0,_)).
site(site_serrage, tcp(0,53000,13200,0,9000,_)).

/*
Programmation XYZ du robot
--------------------------
*/

programme :-
     initialisation_robot,
     tantque( magasins_pas_vides,
            [ saisir(plaque_a),
              deposer_dans(montage),
              saisir(plaque_b),
              deposer_dans(montage, avec , delta(0,0,200,0,0,0)),
              saisir(pince),
              tourner_montage,
              fixer(site_serrage),
              saisir(pince),
              tourner_montage,
              fixer(site_serrage),
              saisir(pince),
              tourner_montage,
              fixer(site_serrage),
              tourner_montage,
              retirer_de(montage, avec, delta(0,0,100,0,0,0)),
              deposer_dans(depot)
            ]),
     initialisation_robot,
fin_programme.

/*
-------------------------------------------------------------------------------
*/

saisir(plaque_a) :-
     coordonnees(plaque_a,Tcp1),
     approche(Tcp1,4500,10000,Tcp2),
     parcourir_tube(tube(tol(10000),vit(400),Tcp2)),
     parcourir_tube(tube(tol(100),vit(40),Tcp1)),
     fermer(force(5,5,8)),
     parcourir_tube(tube(tol(100),vit(400),Tcp2)),!.

saisir(plaque_b) :-
     coordonnees(plaque_b,Tcp1),
     approche(Tcp1,4500,10000,Tcp2),
     parcourir_tube(tube(tol(10000),vit(400),Tcp2)),
     parcourir_tube(tube(tol(100),vit(40),Tcp1)),
     fermer(force(5,5,8)),
     parcourir_tube(tube(tol(100),vit(400),Tcp2)),!.
```

...

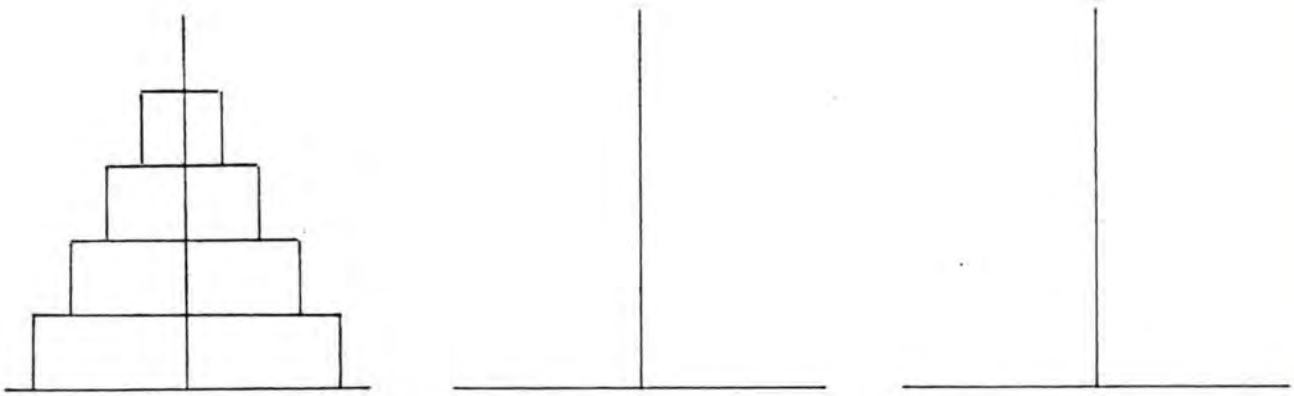
```
saisir(Site) :-
     coordonnees(Site,Tcp1),
     approche(Tcp1,9000,5000,Tcp2),
     parcourir_tube(tube(tol(10000),vit(400),Tcp2)),
     parcourir_tube(tube(tol(100),vit(40),Tcp1)),
     fermer(force(5,5,8)),
     parcourir_tube(tube(tol(100),vit(400),Tcp2)),!.

deposer_dans(Site) :-
     coordonnees(Site,Tcp1),
     approche(Tcp1,9000,5000,Tcp2),
     parcourir_tube(tube(tol(10000),vit(200),Tcp2)),
     parcourir_tube(tube(tol(100),vit(40),Tcp1)),
     ouvrir(force(3,0,2)),
     approche(Tcp1,0,5000,Tcp3),
     parcourir_tube(tube(tol(100),vit(40),Tcp3)),
     fermer(force(0,0,0)),
     ouvrir(force(5,0,8)),!.

deposer_dans(Site, avec, delta(DX,DY,DZ,DT,DR,DL)) :-
     coordonnees(Site,tcp(X,Y,Z,T,R,L)),
     X1 is X + DX,
     Y1 is Y + DY,
     Z1 is Z + DZ,
     T1 is T + DT,
     R1 is R + DR,
     approche(tcp(X1,Y1,Z1,T1,R1,_),9000,6000,Tcp1),
     parcourir_tube(tube(tol(10000),vit(200),Tcp1)),
     parcourir_tube(tube(tol(100),vit(40),tcp(X1,Y1,Z1,T1,R1,_))),
     ouvrir(force(3,0,2)),
     approche(tcp(X1,Y1,Z1,T1,R1,_),0,5000,Tcp2),
     parcourir_tube(tube(tol(100),vit(40),Tcp2)),
     fermer(force(0,0,0)),
     ouvrir(force(5,0,8)),!.

retirer_de(Site, avec, delta(DX,DY,DZ,DT,DR,DL)) :-
     coordonnees(Site,tcp(X,Y,Z,T,R,L)),
     X1 is X + DX,
     Y1 is Y + DY,
     Z1 is Z + DZ,
     T1 is T + DT,
     R1 is R + DR,
     approche(tcp(X1,Y1,Z1,T1,R1,_),0,5000,Tcp1),
     parcourir_tube(tube(tol(10000),vit(200),Tcp1)),
     parcourir_tube(tube(tol(100),vit(40),tcp(X1,Y1,Z1,T1,R1,_))),
     fermer(force(5,5,8)),
     approche(tcp(X1,Y1,Z1,T1,R1,_),9000,5000,Tcp2),
     parcourir_tube(tube(tol(100),vit(40),Tcp2)),!.

fixer(Site) :-
     coordonnees(Site,Tcp1),
     approche(Tcp1,0,3000,Tcp2),
     parcourir_tube(tube(tol(10000),vit(400),Tcp2)),
     parcourir_tube(tube(tol(50),vit(40),Tcp1)),
     ouvrir(force(7,0,8)),
     parcourir_tube(tube(tol(100),vit(400),Tcp2)),!.

tourner_montage :-
     write('Veuillez tourner la station de montage de 90 degres svp'),
     get0(_),nl.
```

...

```
magasins_pas_vides :-
     write('reste - t il encore des plaques dans les magasins des stations '),
     read(oui).


coordonnees(X,C) :- site(X,C).


/*
-----------------------------------------------------------------------------

Implementatation des structures de controle
-------------------------------------------
*/

tantque(X,Y) :-
     tester_cond(X),
     exec(Y),
     fail.

tantque(_,_).

exec([]):-!.
exec([X|Y]) :-
     call(X),!,
     exec(Y).


tester_cond(X) :- not call(X),!,fail.
tester_cond(X).
tester_cond(X) :- tester_cond(X).

ctr_occurence([]).

fin_programme.

?- consult(interface_xyz).
```

## 6.2. RESOLUTION DU PROBLEME DES TOURS DE HANOI PAR UN ROBOT

Le problème des tours de Hanoï est un jeu de trois pôles et d'un ensemble de disques. Les disques sont tous de diamètres différents. Au début du jeu, les disques sont empilés sur le pôle gauche de manière à former une tour. Lorsqu'un disque est empilé sur un autre disque, le disque qui sert de base doit toujours être de diamètre supérieur à celui du disque qui repose sur lui.



Pôle de gauche  Pôle du centre  Pôle de droite

Situation initiale du jeu

Le jeu consiste à déplacer la tour du pôle de gauche vers le pôle de droite. On ne peut déplacer qu'un seul disque à la fois, et il faut toujours respecter la règle d'empilement au cours du déroulement du jeu.

L'algorithme des tours de Hanoï n'est pas une nouveauté. La nouveauté de ce programme réside dans le fait qu'il est destiné à piloter un robot en vue de résoudre le problème des tours de Hanoï de manière automatique.

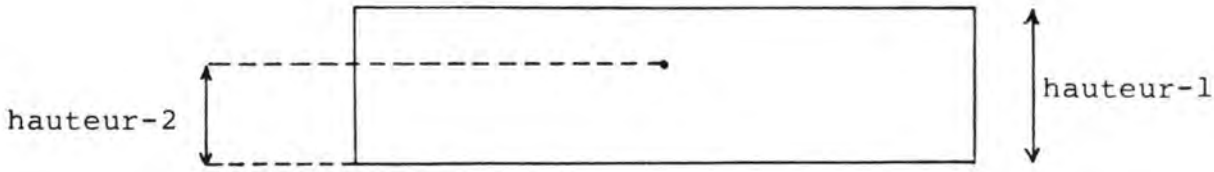
Le programme codé en Prolog permet de résoudre le problème avec n disques.

...

6.2.1. description des disques

Pour chaque disque il y a une description.

plateau (nom, hauteur-1, hauteur-2)

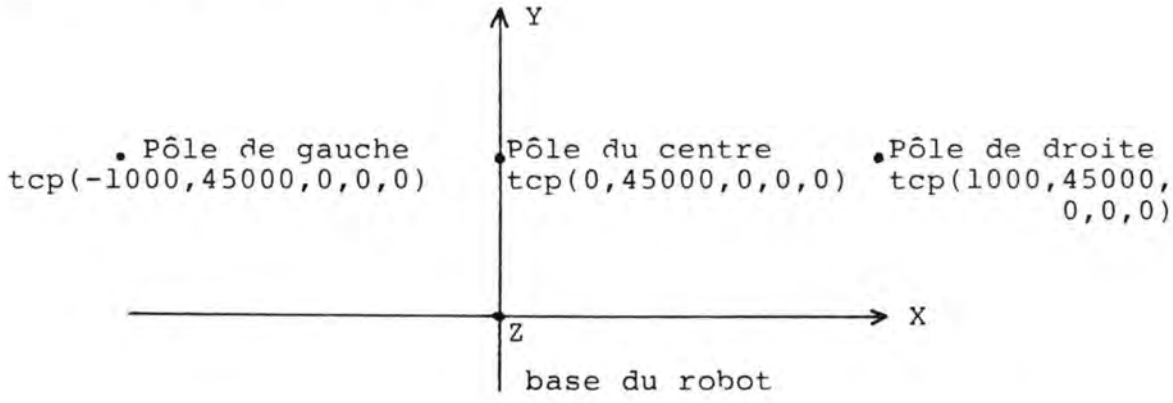



hauteur-1 : donne la hauteur du disque;
hauteur-2 : donne la hauteur à laquelle il faut saisir le disque pour le déplacer;
nom : permet l'identification du plateau.

Les disques doivent être définis par ordre décroissant de diamètres.

6.2.2. description des pôles

Les pôles sont fictifs dans notre cas. Ils sont simplement remplacés par une description d'une position XYZ fixe dans l'espace de travail du robot.




6.2.3. calcul des coordonnées du TCP de saisie du plateau situé au sommet d'une tour

Lorsque l'on désire déplacer un plateau du sommet d'une tour, il faut calculer son tcp de saisie.

Les plateaux sont empilés au-dessus d'un pôle. Donc, la position en X, Y, le lacet, le roulis et le tangage du site de saisie du plateau du sommet restent inchangés. Seule la coordonnée Z change et dépend de la hauteur de la tour.

...

- Il suffit donc de faire la somme des hauteurs-1 de tous les plateaux situés en-dessous du plateau;
- On ajoute la hauteur-2 du plateau situé au sommet et on obtient ainsi la coordonnée Z du tcp de saisie.

6.2.4. calcul des coordonnées du TCP de départ d'un plateau au sommet d'une tour

La méthode est semblable à celle décrite ci-dessus.

- Calcul de la somme des hauteurs-1 de tous les plateaux empilés sur le pôle:
- On y ajoute la hauteur-2 du plateau à déplacer et on obtient la coordonnée Z du tcp de départ du plateau.

```
/*
PROBLEME DES TOURS DE HANOI
---------------------------

*/

plateau(diam-3,750,100).
plateau(diam-2,700,100).
plateau(diam-1,780,100).
plateau(diam-0,700,100).

pgm :- initialisation_robot,
       retractall(instr),
       letsgo,
       assert(instr(end)),
       repeat,
          instr(X),
          appel(X) ,
       X == end,!,
       initialisation_robot.

appel(end).
appel(X) :- call(X).

letsgo :-      retractall(site(_,_,_)),
               assert(site('piquet de droite',tcp(28000,0,0,9000,0,_),[])),
               assert(site('piquet du centre',tcp(21500 ,18000,0,9000,0,_),[])),
               findall(X,plateau(X,_,_),P),
               assert(site('piquet de gauche',tcp( 4800 ,27500,0,9000,0,_),P)),
               long(P,L),!,
               hanoi(L).

hanoi(N) :- move(N,'piquet de gauche','piquet du centre','piquet de droite').
move(0,_,_,_):- !.
move(N,A,B,C) :-
     N > 0,
     M is N - 1,
     move(M,A,C,B),
     inform(A,B),
     move(M,C,B,A),!.

inform(X,Y) :-
     write('deplacer un disque du '),
     write(X),
     write(' vers le '),
     write(Y),nl,
     mouvement(X,Y),
     deplace(X,Y),!.

mouvement(Site_origine,Site_destination) :-
     plateau_a_deplacer(Site_origine,Plateau),
     debut_approche(Site_origine,Tube1),
     approche_saisie(Site_origine,Plateau,Tube2),
     eloignement(Site_origine,Tube3),
     debut_approche(Site_destination,Tube4),
     approche_depot(Site_destination,Plateau,Tube5),
     eloignement(Site_destination,Tube6),
     assert(instr(parcourir_circ(circ([Tube1,Tube2])))),
     assert(instr(fermer(force(7,5,5)))),
     assert(instr(parcourir_circ(circ([Tube3,Tube4,Tube5])))),
     assert(instr(ouvrir(force(7,5,5)))),
     assert(instr(parcourir_tube(Tube6))),!.
```

...

```
plateau_a_deplacer(Site_origine,plateau(Nom,H1,H2)):-
     site(Site,_,[Nom|R]),
     plateau(Nom,H1,H2),!.

deplace(Site_origine,Site_destination) :-
     retract(site(Site_origine,Tcp1,[S1|P1])),
     retract(site(Site_destination,Tcp2,P2)),
     assert(site(Site_destination,Tcp2,[S1|P2])),
     assert(site(Site_origine,Tcp1,P1)),!.

debut_approche(Site,tube(tol(1000),vit(400),tcp(X,Y,Z1,T,R,L))):-
     site(Site,tcp(X,Y,Z,T,R,L),_),
     Z1 is Z + 10000,!.

approche_saisie(Site, plateau(_,H1,H2), tube(tol(20),vit(400),tcp(X,Y,Z1,T,R,L))):-
     coord_sommet_site(Site,tcp(X,Y,Z,T,R,L)),
     Z1 is Z - H1 + H2,!.

approche_depot(Site, plateau(_,_,H2), tube(tol(20),vit(400),tcp(X,Y,Z1,T,R,L))):-
     coord_sommet_site(Site,tcp(X,Y,Z,T,R,L)),
     Z1 is Z + H2,!.

eloignement(Site,tube(tol(100),vit(400),tcp(X,Y,Z1,T,R,L))):-
     site(Site,tcp(X,Y,Z,T,R,L),_),
     Z1 is Z + 10000,!.

coord_sommet_site(Nom_site,tcp(X,Y,Z,T,R,L)) :-
     site(Nom_site,tcp(X,Y,Z1,T,R,L),C),
     somme_hauteur_plateaux(C,Somme),
     Z is Z1 + Somme,!.

somme_hauteur_plateaux([],0).
somme_hauteur_plateaux([Plateau|P],S) :-
     plateau(Plateau,H,_),
     somme_hauteur_plateaux(P,N),
     S is N + H,!.

long([],0).
long([X|Y],L):- long(Y,N),L is N + 1.

findall(O,L,_):-asserta(found(mark)),
     call(L),
     asserta(found(O)),
     fail.

findall(_,_,S):-collectfound([],M),
     !,S=M.

collectfound(L,S):-getnext(O),!,
     collectfound([O|L],S).
collectfound(S,S).

getnext(O):-     retract(found(O)),!,
     O\==mark.

?- reconsult(interface_xyz).
```

...

## 6.3. DEFINITION D'UNE TACHE DE PALETISATION

La palétisation consiste à saisir une série d'objets et à les déplacer côte à côte en ligne et, ou, en colonne.

L'application suivante est une application de palétisation.

Une presse fabrique une série de petits objets. On demande au robot de saisir ces objets et de les placer dans un bac métallique selon trois lignes et quatre colonnes.

Avant toute nouvelle exécution, le robot doit se réinitialiser pour éviter tout dérèglement du bras articulé et garantir ainsi la précision des mouvements (prédicat initialisation-robot).

Après la première initialisation du robot, on répète une séquence d'instructions.

- Attente d'un signal de synchronisation : le robot doit être synchronisé avec la presse pour éviter la destruction du bras par la presse. Pour l'instant, nous nous contenterons d'utiliser n'importe quelle touche du clavier en guise de signal de synchronisation;
- Sortir l'objet de la presse : lorsque la presse est relevée l'objet fabriqué est accessible. La presse présente cet objet toujours de la même manière (il suffit simplement d'amener la pince à la position de l'objet et de la serrer) (prédicat sortir-de-presse);
- Déposer dans le bac l'objet saisi dans la presse (prédicat déposer-dans-bac) : cet objet doit être aligné à côté des objets déjà déposés dans le bac. La position de dépôt est différente pour tous les objets. Il faut donc calculer cette position avant de déposer l'objet (prédicat position-cible). Lorsque l'objet est dans le bac, la pince doit être ouverte très largement pour ne pas bousculer l'objet (éventuel) se trouvant juste à côté de l'objet déposé. Mais cette pince doit être ouverte suffisamment pour permettre de libérer l'objet de la pince. On aura donc recours à l'utilisation d'une force très faible et très brève pour l'ouverture de cette pince (prédicat ouvrir (force(2,1,1))).

Cette séquence d'instructions se répète jusqu'à ce que le nombre de rangées et de colonnes d'objets alignés dans le bac corresponde au nombre de rangées et de colonnes désirées (prédicat bac-rempli)

...

Programme de paletisation

```
position_presse(tcp(37315,22043,6728,6000,0,_)).
repere_bac(tcp(~10000,28000,1600,9000,0,18000)).
nombre_de_pieces(0).
dx(2400).
dy(2000).
pieces_par_rang(3).
nombre_de_rangs(4).

paletisation :-
          initialisation_robot,
          initialisation,
          repeat,
               synchro_presse,
               sortir_de_presse,
               deposer_dans_bac,
          bac_rempli,!,
          initialisation_robot.

/*-------------------------------------------------------------------------------*/
initialisation :-
          position_presse(T),
          approche(T,6000,4000,U),
          approche(T,6000,4000,V),
          assert(devant_presse(U)),
          assert(approche_presse(V)),
          immediat(dep(vit(400),U)),
          fermer(force(0,0,0)),
          ouvrir(force(5,3,10)),
          fermer(force(6,0,4)),
          ouvrir(force(0,0,0)),!.


sortir_de_presse :-
          position_presse(P),
          approche_presse(U),
          parcourir_circ(circ([tube(tol(100),vit(400),U),
                               tube(tol(100),vit(400),P)])),
          fermer(force(2,1,15)),!.

deposer_dans_bac :-
          position_cible(P),
          approche_presse(U),
          devant_presse(R),
          preparer_approche(P,6000,S),
          parcourir_circ(circ([tube(tol(100),vit(400),U),
                               tube(tol(1000),vit(400),S),
                               tube(tol(100),vit(400),P)])),
          ouvrir(force(2,1,1)),
          parcourir_circ(circ([tube(tol(100),vit(400),S),
                               tube(tol(1000),vit(400),R)])),!.


synchro_presse :- write('Signal de synchronisation ) '),get0(X),nl.
```



...

```
bac_rempli :-
          nombre_de_pieces(NP),
          nombre_de_rangs(NR),
          pieces_par_rang(PR),
          NP =:= NR * PR.


/*-------------------------------------------------------------------------------*/

position_cible(tcp(X,Y,PZ,9000,R,_)) :-
          repere_bac(tcp(PX,PY,PZ,PT,PR,PL)),
          dx(DX),
          dy(DY),
          pieces_par_rang(NR),
          nombre_de_pieces(NT),
          X is PX + (NT / NR) * DX,
          Y is PY + (NT - (NT / NR) * NR) * DY,
          incrementer_total_pieces,
          notrace,
          transformation_inverse(tcp(X,Y,PZ,9000,0,_),coord(L,_,_,_,_)),
          R is PL - L,!.

preparer_approche(tcp(X,Y,Z,T,R,_),Dist,tcp(X,Y,Z1,T,R,_)) :-
          Z1 is Z + Dist.

incrementer_total_pieces :-
          retract(nombre_de_pieces(NP)),
          N is NP + 1,
          assert(nombre_de_pieces(N)),!.

?-consult(interface_xyz).
```

## 6.4. REALISATION D'UN BOITIER D'APPRENTISSAGE

L'objectif du module d'apprentissage est de permettre à l'utilisateur de programmer rapidement le robot RM 501.

- Ce module d'apprentissage permet à la fois un apprentissage au niveau XYZ et un apprentissage au niveau articulaire. On met à la disposition de l'utilisateur une série de commandes directement activées à l'aide du clavier.
- L'écran du micro ordinateur affiche toujours les cinq valeurs des variables articulaires
  la position cartésienne et l'orientation du bout de la pince;
  la vitesse actuelle de translation de la pince:
  la valeur du pas de translation et de rotation.
- Grâce à l'affichage de la position cartésienne de la pince et de son orientation, on peut aussi utiliser ce boîtier d'apprentissage comme outil de mesure de la position et de l'orientation des objets de l'espace de travail. Il suffit simplement de faire coïncider la position de la pince du robot avec la position des objets disposés dans l'espace de travail.
- Ce module permet l'enregistrement d'un programme acquis par apprentissage ainsi que sa réexécution.

### 6.4.1. Mouvements cartésiens de la pince (niveau XYZ)

| Touche | | Fonction activée |
|---|---|---|
| D | -----→ | X := X + Pas-translation |
| H | -----→ | X := X - Pas-translation |
| T | -----→ | Y := Y + Pas-translation |
| V | -----→ | Y := Y - Pas-translation |
| F | -----→ | Z := Z + Pas-translation |
| G | -----→ | Z := Z - Pas-translation |
| Y | -----→ | Tangage := tangage + Pas-rotation; Les positions XYZ restent inchangées; |
| N | -----→ | Tangage := tangage - Pas-rotation; Les positions XYZ restent inchangées; |
| Z | -----→ | Roulis := roulis + Pas-rotatation; Les positions XYZ restent inchangées; |

...

X -----→ Roulis := roulis - Pas-rotation;
Les positions XYZ restent inchangées;

6.4.2. Mouvements par actionnement des articulations (niveau articulaire)

| Touche | | Fonction activée |
|---|---|---|
| 1 | -----→ | Angle de rotation du corps:= Angle de rotation du corps + Pas-rotation; Les positions XY sont modifiées en conséquence; |
| 2 | -----→ | Angle de rotation du corps := Angle de rotation du corps - Pas-rotation; Les positions XY sont modifiées en conséquence: |
| Q | -----→ | Angle de rotation de l'épaule := Angle de rotation de l'épaule + Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| W | -----→ | Angle de rotation de l'épaule := Angle de rotation de l'épaule - Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| A | -----→ | Angle de rotation du coude : = Angle de rotation du coude + Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| S | -----→ | Angle de rotation du coude := Angle de rotation du coude - Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| Z | -----→ | Angle de tangage du poignet := Angle de tangage du poignet + Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| X | -----→ | Angle de tangage du poignet := Angle de tangage du poignet - Pas-rotation; Les positions XYZ sont modifiées en conséquence; |
| Z | -----→ | Angle de roulis du poignet := Angle de roulis du poignet + Pas-rotation; Les positions XYZ restent inchangées; |
| X | -----→ | Angle de roulis du poignet := Angle de roulis du poignet - Pas-rotation; Les positions XYZ restent inchangées; |

...

6.4.3. Ouverture et fermeture de la pince

- -----→ Fermeture de la pince;
= -----→ Ouverture de la pince;

6.4.4. Enregistrement et exécution des mouvements

ret ou space -----→ enregistrement de la position actuelle;
* -----→ Exécution du programme enregistré.

6.4.5. Mise à jour des variables du système

I -----→ Vitesse := vitesse + 40 mm/s
O -----→ Vitesse := vitesse - 40 mm/s
K -----→ Mise à jour du Pas utilisé pour les déplacements en XYZ
' -----→ Mise à jour du pas de rotation des articulations.

6.4.6. Contrôle de l'exécution du programme "teacher"

Ctrl A -----→ Terminaison du programme
Ctrl ₀ -----→ Avortement du programme

6.4.7. Implémentation du boîtier d'apprentissage

Etant donné la longueur et la simplicité du programme, son listing source ne figurera pas ci-dessous. Toutefois, ce listing se retrouve en Annexe E de ce travail.

...

# 7. LA COUCHE DE NIVEAU OBJET

L'usage d'un robot est naturellement concerné par le positionnement de la pince et le déplacement des objets. Jusqu'à présent, nous n'avions manipulé que des tcp. Ces tcp sont des informations pauvres car ils ne représentent uniquement qu'un point de saisie des objets, mais pas leur position réelle, leur taille, etc ...

Lorsqu'un objet était déplacé, la mise à jour de ses repères devait être assurée par le programme de l'utilisateur. La manière d'approcher les objets, la force que la pince doit exercer sur les objets sont des caractéristiques de ces mêmes objets. Jusqu'à présent, ces caractéristiques étaient dissociées des objets. C'est encore le programme de l'utilisateur qui devait gérer ces caractéristiques.

En outre, nous avons jusqu'à présent considéré que tous les objets étaient indépendants. En effet, on associait un tcp à chaque objet et tout tcp était modifié indépendamment des autres. Cette manière de voir les choses n'est pas très réaliste car en réalité, certains objets peuvent être solidaires d'autres.

Ainsi, lorsqu'on effectue un montage (utilisation très fréquente d'un robot), tous les objets qui le constituent sont solidaires les uns des autres. Cela signifie que si on déplace le montage, on déplace par la même occasion tous ses objets. Il faudra donc mettre à jour la description de tous ces objets.

La couche de niveau "objet" remédie à ces faiblesses en offrant de nouvelles primitives de programmation. A chaque objet de l'espace de travail est associé un descripteur contenant toutes ses caractéristiques. L'ensemble de ces descripteurs constitue un modèle de l'espace de travail du robot. Ce modèle sera utilisé par le planificateur de mouvements de la couche supérieure.

La couche "XYZ" permettra également d'exprimer explicitement les liaisons entre les objets de l'espace de travail. On utilisera à cet effet des primitives de liaison définies dans la couche.

Le rôle de la couche "objet" sera d'offrir des primitives de mise à jour du modèle de l'espace de travail. Ces primitives puissantes permettront d'améliorer, par la même occasion, la lisibilité des programmes. Par exemple, les objets sont référencés par leur nom et plus par un tcp fort peu parlant.

## 7.1. Les méta-objets manipulés dans la couche "objet"

Pour éviter la confusion entre objet figurant dans l'espace de travail du robot et objet défini par des compétences dans le mode de structuration adopté dans ce travail, nous désignerons celui-ci sous le nom de "méta-objet" lorsqu'une ambiguïté est possible.

Tous les méta-objets définis dans l'interface de la couche "XYZ" sont présents dans la couche "objet". Mais cette dernière contient également une série de concepts qui lui sont propres.

Il existe des relations structurelles entre les méta-objets de la couche de niveau "objet". Ces relations sont exprimées par la figure 7.1.

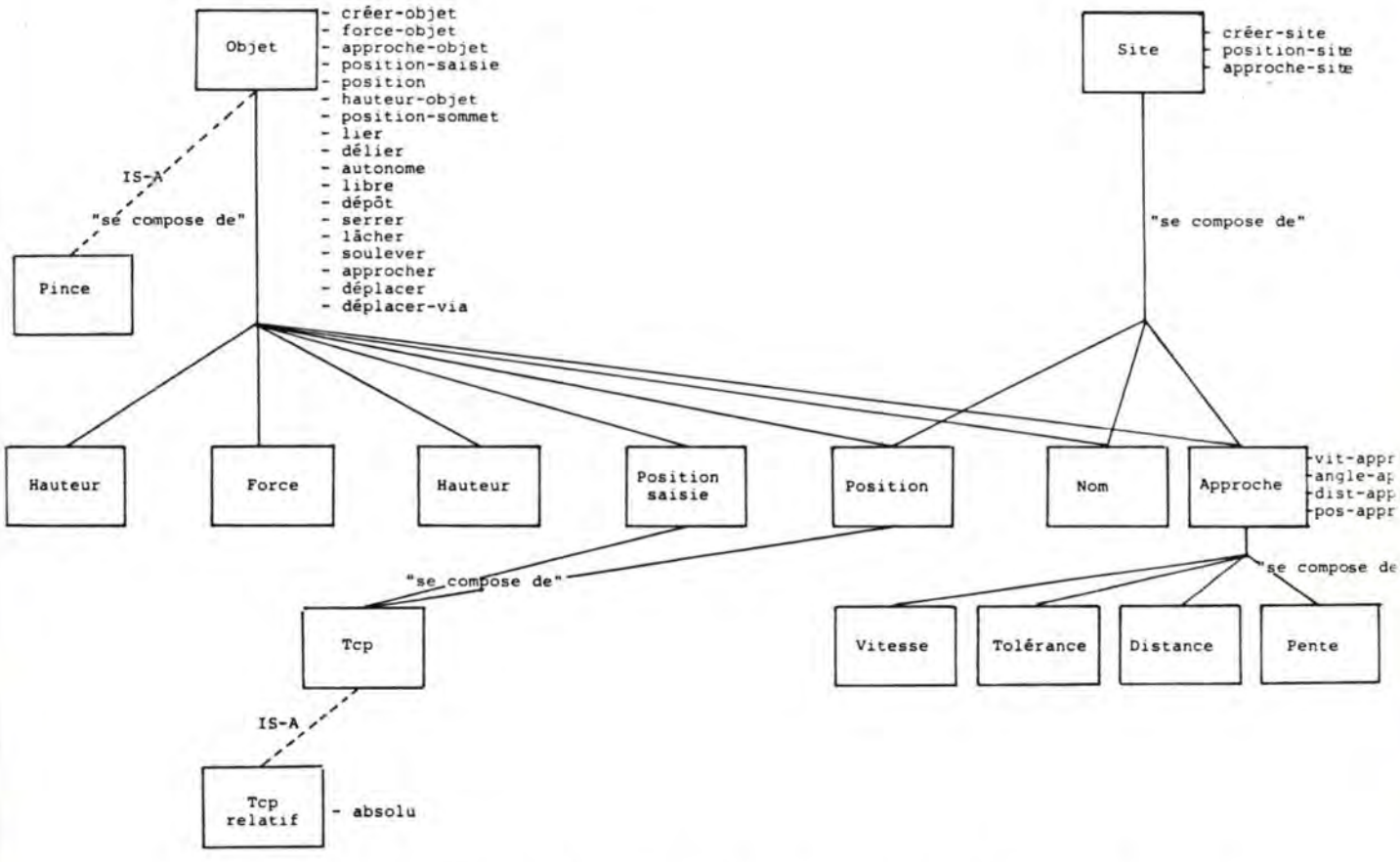



Fig. 7.1. Structuration des méta-objets définis dans la couche "objet"

...

Le concept d'approche : le méta-objet "approche" permet de calculer une position intermédiaire d'approche (ou d'éloignement) par laquelle devra passer la trajectoire du tcp de la pince pour saisir (ou pour s'éloigner ou soulever) un objet de l'espace de travail. Le concept de point d'approche a déjà été abordé au paragraphe 5.2.4.1. (compétence "approche" du tcp). Le méta-objet "approche" est défini à l'aide des concepts suivants :

- la_vitesse : détermine la vitesse d'approche ou d'éloignement du centre de la pince.
- la_tolérance : détermine l'écart maximal de la trajectoire du centre de la pince par rapport à une trajectoire théorique rectiligne d'approche.
- la_distance : détermine la distance de la trajectoire théorique rectiligne d'approche.
- la_pente : détermine la pente de la droite qui représente la trajectoire rectiligne théorique d'approche. Cette pente est exprimée par rapport aux axes du repère $R_o$ de l'espace de travail.

Ces concepts permettent de construire un tube délimitant la trajectoire d'approche (ou d'éloignement).

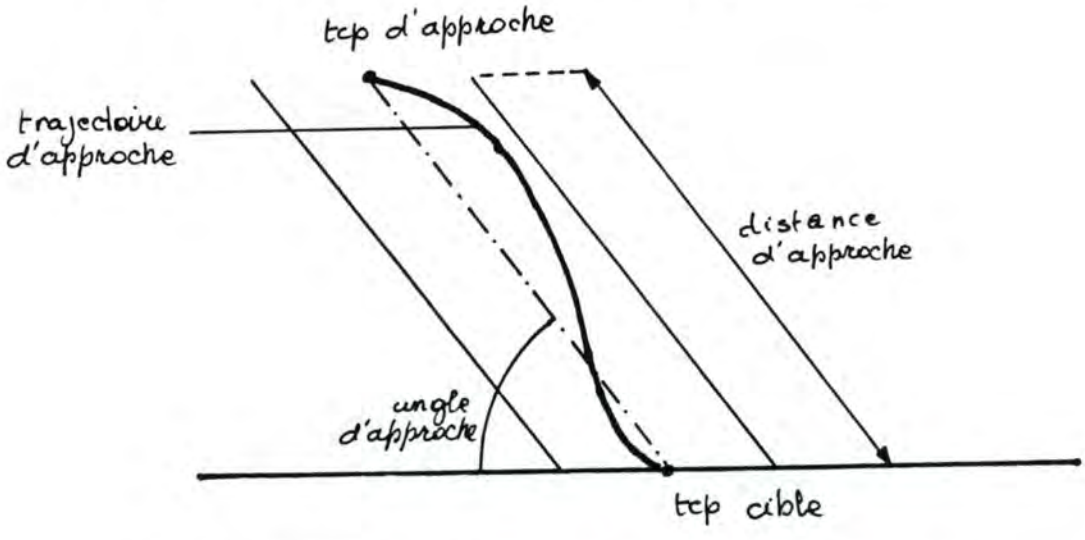



Fig. 7.2 - Illustration du tube d'approche (et d'éloignement)

...

Le concept de site : le site représente une position fixe de l'espace de travail. Il est défini à l'aide des informations suivantes :

- le_nom : identifie le site de manière univoque.
- la_position : représente à l'aide d'un tcp, la position fixe de l'espace de travail.
- l'approche : détermine la manière d'approcher la position fixe (voir concept d'approche).

Le concept d'objet : le concept d'objet permet de décrire les caractéristiques des objets disposés dans l'espace de travail du robot. Le concept d'objet est défini à l'aide des concepts suivants :

- le_nom : identifie de manière univoque l'objet disposé dans l'espace de travail.
- la_position_de_saisie : est un tcp qui représente un repère associé à la position de préhension de l'objet de l'espace de travail.
- la_position : identifie la position et l'orientation de l'objet de l'espace de travail par rapport au repère $R_o$. Cette orientation et cette position sont représentés à l'aide d'un repère associé à la base de l'objet. Ce repère est représenté à l'aide d'un tcp.
- l'approche : contient les informations nécessaires à la construction d'une trajectoire d'approche (ou d'éloignement) de la position de saisie de l'objet.
- la_force : désigne la préssion que doit exercer la pince sur l'objet pour le saisir et le déplacer. Cette force est choisie en fonction du poids et de la solidité de l'objet.
- la_hauteur : représente la hauteur de l'objet.
- le_rayon : représente le rayon du cylindre de révolution de l'objet autour d'un axe de symétrie vertical perpendiculaire à la base sur laquelle il repose.

Le concept de pince : la pince est un méta-objet qui représente tout ce qui subsiste du robot au niveau "objet". Seuls les mouvements de la pince nous intéressent car en définitive, c'est elle qui manipule les objets de l'espace de travail.

...

Déjà au niveau "XYZ", nous avions recours à cette hypothèse simplificatrice où l'extrémité de l'outil, exprimée par un tcp, était notre centre d'intérêt. La pince est également un objet de l'espace de travail, il est donc représenté par un méta-objet identique à ceux utilisés pour définir les autres objets. Il faut noter toutefois quelques particularités :

- le_nom : on utilisera le mot"pince" pour désigner la pince. L'utilisateur doit donc veiller à ne pas appeler "pince" un autre objet de l'espace de travail.
- la_force : sera toujours nulle car la pince ne peut se serrer elle-même.
- l'approche : la vitesse, la tolérance, la distance et la pente de l'approche seront tous nuls car la pince ne peut s'approcher elle-même.
- la_position : représentera la position et l'orientation du repère associé à l'extrémité de la pince.
- la_position_de_saisie : identique à la position.

Les autres concepts sont identiques à ceux des meta-objets représentant les objets de l'espace de travail.

Le concept de tcp relatif : ce concept est nouveau mais n'est pas nécessairement inhérant à la couche objet. Jusqu'à présent, toutes les positions étaient exprimées par rapport au repère $R_o$ de l'espace de travail. Tous les objets de l'espace de travail étaient toujours indépendants car les tcp qui les représentaient l'étaient également. Au niveau objet, nous avons émis l'hypothèse que des objets de l'espace de travail pouvaient être liés. Il faut représenter ces liaisons au niveau des repères associés aux objets. On peut donc exprimer la solidarité entre les objets en remplaçant leurs repères relatifs exprimés dans un repère de l'un des objets liés. On pourrait exprimer par exemple les repères de tous les objets liés dans le repère associé à la position de saisie de l'un des objets. On déplacera simultanément l'ensemble de tous les objets liés, en déplaçant l'objet dont le repère de saisie sert de référence.

...

## 7.2. Compétences des méta-objets définis dans la couche de niveau objet

### 7.2.1. Compétences du méta-objet "approche"

VITESSE-APPR

Méta-objet créé : vitesse
Effet : donne la vitesse de l'approche.

ANGLE-APPR

Méta-objet créé : pente
Effet : donne la pente de la trajectoire théorique rectiligne de l'approche.

TOLERANCE-APPR

Méta-objet créé : tolérance
Effet : donne la tolérance de l'approche.

DISTANCE-APPR

Méta-objet créé : distance
Effet : donne la distance de l'approche.

POSITION-APPR

Méta-objet requis : tcp
Méta-objet créé : tcp
Effet : calcule et renvoie un tcp d'approche du tcp donné comme argument.
La pente par rapport à $R_o$ de la droite passant par l'origine des repères représentés par les deux tcp, est donnée par la pente de l'objet de classe "approche" dont on a activé la compétence "position intermédiaire".
La distance séparant l'origine des deux repères est également donnée par l'objet de classe "approche". L'axe $X_i$ des deux repères se situent toujours dans le même plan vertical.
Méthode : - active les compétences "angle-appr" et "dist-appr" de l'objet "approche" pour obtenir la pente et la distance d'approche.

...

- active la compétence "approche" du tcp donné comme argument. La compétence "approche" du tcp est définie dans la couche "XYZ". Lors de l'activation de la compétence "approche", on fournit comme argument la pente et la distance d'approche déterminés précedemment.

APPR

Méta-objet créé : approche
Méta-objet requis : vitesse, tolérance, pente, distance.
Effet : crée une instance d'objet "approche" à l'aide des concepts de vitesse, de tolérance, de pente et de distance donnés comme arguments.

7.2.2. Compétences des méta-objets "site"

CREER-SITE

Méta-objet créé : instance de site
Méta-objet requis : nom de site, tcp, approche.
Effet : crée une instance de site en utilisant les objets donnés comme argument.

POSITION-SITE

Méta-objet créé : tcp
Méta-objet requis : nom de site
Effet : donne le tcp de l'objet de classe "site" identifié par le nom donné en argument.

APPROCHE-SITE

Méta-objet créé : approche
Méta-objet requis : nom de site
Effet : renvoie l'objet approche qui caractérise l'objet de classe "site" identifié par le nom de site donné comme argument.

...

### 7.2.3. Compétences des méta-objets "objet"

CREER-OBJET

Méta-objet créé : méta-objet de classe "objet"
Méta-objet requis : nom d'objet, approche, force, rayon, position, position-saisie.
Effet : crée une instance de méta-objet "objet" à l'aide de méta-objets donnés comme arguments. L'objet de l'espace de travail et le méta-objet qui le représente sont identifiés univoquement par leur nom d'objet.

FORCE-OBJ

Méta-objet créé : force
Méta-objet requis : nom d'objet
Effet : donne la force que doit exercer la pince sur l'objet représenté par le méta-objet "objet" pour assurer son déplacement. L'objet et le méta-objet sont identifiés par le nom donné comme argument.

APPROCHE-OBJ

Méta-objet créé : approche
Méta-objet requis : nom d'objet
Effet : renvoie le méta-objet "approche" qui caractérise le méta-objet identifié par le nom d'objet donné comme argument.

POSITION-SAISIE

Méta-objet créé : tcp
Méta-objet requis : nom d'objet
Effet : donne la position de saisie de l'objet. Cette position est contenue dans le méta-objet de classe "objet" qui représente cet objet. L'objet et le méta-objet sont identifiés par le nom donné comme argument.

POSITION

Méta-objet créé : tcp
Méta-objet requis : nom d'objet
Effet : donne la position de la base de l'objet. Cette position est contenue dans le méta-objet identifié par le nom d'objet donné comme argument.

HAUTEUR-OBJ

Méta-objet créé : hauteur
Méta-objet requis : nom d'objet
Effet : donne la hauteur de l'objet représenté par un méta-objet de classe "objet", identifié par le nom d'objet donné comme argument.

POSITION-SOMMET

Méta-objet créé : tcp
Méta-objet requis : nom d'objet
Effet : calcule et renvoie la position du sommet de l'objet. Les informations (position et hauteur) permettant le calcul de la position du sommet sont contenues dans le méta-objet de classe "objet" identifié par le nom donné comme argument.

LIER

Méta-objet requis : deux noms d'objet
Effet : a pour effet de rendre solidaires les objets de l'espace de travail dont on a donné les noms. Ces noms permettront aussi d'accéder aux méta-objets qui décrivent ces objets. Toute mise à jour de la position de l'un des objets entraînera la mise à jour de l'autre.

DELIER

Méta-objet requis : deux noms d'objet
Effet : rend autonome les deux objets liés dont on a donné le nom.

...

AUTONOME

Méta-objet créé : booléen
Méta-objet requis : nom d'objet
Effet : vérifie l'autonomie de l'objet dont on a donné le nom. Si l'objet n'est pas lié à un autre objet, la compétence "autonome" renvoie la valeur "vrai", sinon la valeur "faux".

LIBRE

Méta-objet créé : booléen
Méta-objet requis : nom d'objet
Effet : verifie s'il n'y a pas d'autre objet posé sur l'objet dont on a donné le nom.

DEPOT

Méta-objet créé : site
Méta-objet requis : nom d'objet 1, nom d'objet 2
Effet : calcule et renvoie un objet de classe "site" qui représente la position de l'espace de travail que doit atteindre la pince du robot pour placer l'objet 1 sur l'objet 2. Ces objets sont respectivement identifiés par nom d'objet 1 et nom d'objet 2.

SERRER

Méta-objet requis : nom d'objet
Effet : provoque la fermeture de la pince avec une force donnée par le méta-objet de classe "objet" identifié par le nom d'objet donné en argument.

LACHER

Meta-objet requis : nom d'objet
Effet : provoque l'ouverture de la pince avec une force donnée par le méta-objet de classe "objet" identifié par le nom d'objet donné en argument.

...

SOULEVER

Méta-objet requis : nom d'objet
Effet : provoque le parcours d'un tube construit à l'aide des informations contenues dans le méta-objet "approche" qui caractérise le méta-objet identifié par le nom d'objet donné comme argument. Ce tube représentera une trajectoire d'éloignement du tcp de la pince permettant de soulever l'objet serré par la pince.
Méthode : - active la compétence "position-saisie" du méta-objet identifié par le nom d'objet donné comme argument.
- active la compétence "position-approche" du méta-objet identifié par le nom d'objet donné comme argument. Lors de l'activation de la compétence "position-approche", on donne pour argument le tcp obtenu antérieurement.
- on parcourt le tube construit à l'aide de la vitesse et de la tolérance caractérisant le méta-objet "approche" et à l'aide du dernier tcp calculé.

APPROCHER

Méta-objet requis : nom d'objet
Effet : provoque le parcours d'un tube construit à l'aide des informations contenues dans le méta-objet "approche" du méta-objet identifié par le nom d'objet donné comme argument. Ce tube représentera une trajectoire d'approche du tcp de la pince permettant d'approcher la position de saisie de cet objet.

DEPLACER

Méta-objet requis : nom d'objet 1, nom d'objet 2, vitesse, tolérance.
Effet : provoque la saisie et le déplacement

...

de l'objet 1 identifié par le nom d'objet 1. L'objet 1 doit être posé au-dessus de l'objet 2 identifié par le nom d'objet 2. La trajectoire permettant le soulèvement de l'objet 1 est conforme au méta-objet "approche" contenu dans le méta-objet qui décrit l'objet 1. La trajectoire permettant le dépôt de l'objet 1 sur l'objet 2 est conforme au méta-objet "approche" contenu dans le méta-objet qui décrit l'objet 2. La trajectoire proprement dite de déplacement de l'objet 1 est caractérisée par la vitesse et la tolérance données comme arguments.

Méthode : - activer la compétence "approcher" en donnant le nom d'objet 1 comme argument.
- activer la compétence "serrer" en donnant le nom d'objet 1 comme argument.
- activer la compétence "soulever" en donnant le nom d'objet 1 comme argument.
- activer la compétence "dépôt" du méta-objet "objet" identifié par le nom d'objet 2 afin d'obtenir un site représentant la position de dépôt.
- calculer le tcp d'approche du site de dépôt en activant la compétence "position-approche" du tcp du site de dépôt.
- parcourir un tube construit à l'aide des objets vitesse et tolérance donnés comme arguments et à l'aide du tcp d'approche du site de dépôt.
- amener le tcp de la pince sur la position identifiée par le site de dépôt et selon une trajectoire conforme à celle exprimée par le méta-objet "approche" du site de dépôt.
- relacher l'objet 1.

...

- éloigner la pince en parcourant en sens inverse, la dernière trajectoire d'approche du tcp de la pince.

DEPLACER VIA

Méta-objet requis : nom d'objet 1, nom d'objet 2, vitesse, tolérance, circuit.

Effet : provoque la saisie et le déplacement de l'objet 1 identifié par le nom d'objet 1. L'objet 1 doit être posé au-dessus de l'objet 2 identifié par le nom d'objet 2. La trajectoire permettant le soulèvement de l'objet 1 est conforme au méta-objet "approche" contenu dans le méta-objet qui décrit l'objet 1. La trajectoire permettant le dépôt de l'objet 1 sur l'objet 2 est conforme au méta-objet qui décrit l'objet 2. La trajectoire de déplacement proprement dite, est un circuit obtenu en ajoutant un nouveau tube 1 à la fin de la liste des tubes qui composent ce circuit. Ce tube 1 est construit à l'aide des objets vitesse, tolérance donnés comme arguments et à l'aide d'un tcp d'approche du site de dépôt.

Méthode : similaire à la précédente.

DEPLACER

Méta-objet requis : nom d'objet, nom de site, vitesse, tolérance.

Effet : provoque la saisie et le déplacement de l'objet identifié par le nom d'objet donné comme argument. L'objet déplacé doit être posé sur le site identifié par le nom de site donné comme argument. La trajectoire du tcp de la pince permettant le soulèvement de l'objet est conforme au méta-objet "approche"

...

contenu dans le méta-objet décrivant l'objet à déplacer. La trajectoire du tcp de la pince permettant le dépôt de l'objet sur le site est conforme au méta-objet "approche" contenu dans le méta-objet identifié par le nom de site donné comme argument. La trajectoire proprement dite de déplacement de l'objet est caractérisée par la vitesse et la tolérance données comme arguments.

Méthode : similaire à la précédente.

DEPLACER-VIA

Méta-objet requis : nom d'objet, nom de site, vitesse, tolérance, circuit.

Effet : identique au cas précédent mais la trajectoire de déplacement proprement dite diffère quelque peu. Cette trajectoire est un circuit obtenu en ajoutant un nouveau tube (tube 1) à la fin de la liste des tubes qui composent ce circuit. Le tube 1 est construit à l'aide des objets vitesse, tolérance donnés comme arguments et à l'aide d'un tcp d'approche du site de dépôt.

Méthode : similaire à la méthode utilisée précédemment dans le cas de la compétence "déplacer-via.

### 7.2.4. Compétences du méta-objet "pince"

La pince est un cas particulier d'objet de l'espace de travail. Le méta-objet qui décrit cette pince sera donc un cas particulier de méta-objet de classe "objet". Ce méta-objet héritera donc des compétences qui caractérisent les méta-objets de classe "objet". Quelques-une de ces compétences auront, toutefois, une signification particulière pour la pince. Nous passerons ces compétences en revue et nous expliquerons leurs particularités.

...

D'autre part, la pince est caractérisée par des compétences qui lui sont propres. Nous expliquerons également ces compétences.

POSITION

Méta-objet créé : tcp
Méta-objet requis : nom "pince"
Effet : donne la position et l'orientation actuelles du repère associé à l'extrémité de la pince par rapport au repère $R_o$ de l'espace de travail.

DEPLACER

Méta-objet requis : "pince", nom d'objet, vitesse, tolérance.
Effet : cette compétence est héritée des compétences caractérisant les méta-objets de classe "objet". Le déplacement de la pince sur un objet consiste à faire coïncider l'extrémité de la pince avec la position de saisie de l'objet.

DEPLACER

Méta-objet requis : "pince", nom de site, vitesse, tolérance.
Effet : identique à celui du cas précédent mais cette fois, l'extrémité de la pince sera amenée sur la position définie par le méta-objet "site" identifié par le nom de "site".

DEPLACER-VIA

Méta-objet requis : "pince", nom d'objet, vitesse, tolérance, circuit.
Effet : identique au cas précédent mais le tcp de la pince parcourt un circuit donné comme argument avant de coïncider avec la position de saisie de l'objet identifié par le nom d'objet.

DEPLACER-VIA

Méta-objet requis : "pince", nom de site, vitesse, tolérance, circuit.

...

Effet : identique au cas précédent mais la pince coïncidera avec la position définie par le méta-objet "site" identifié par le nom de "site".

INITIALISATION-PINCE

Effet : provoque la (ré)initialisation du hard-ware du robot. Cette (ré)initialisation entraîne le déplacement de la pince. Il faudra donc mettre à jour la position de cette pince. L'initialisation est donc une compétence de l'objet pince.

7.2.5. Compétences des méta-objets "tcp"

RELATIF

Méta-objet créé : tcp relatif
Méta-objet requis : tcp, tcp repère
Effet : calcule et renvoie le tcp relatif d'un tcp donné par rapport à un autre tcp.

ABSOLU

Méta-objet créé : tcp absolu
Méta-objet requis : tcp, tcp repère
Effet : calcule et renvoie le tcp absolu d'un tcp relatif donné par rapport à un autre tcp repère.

Ces deux compétences permettront de faciliter la mise à jour de la base de données des objets. Lorsque deux tcp sont solidaires, on exprime la position relative de l'un par rapport à l'autre, on déplace le repère représenté par le tcp absolu de ce dernier, on met à jour ce tcp absolu de manière à ce qu'il identifie la nouvelle place du repère qu'il représente et enfin, on calcule le nouveau tcp absolu du premier à partir de son tcp relatif exprimé par rapport à la nouvelle valeur du tcp absolu "repère".

...

## 7.3 Interface de la couche du niveau objet

L'ensemble des méta-objets définis dans la couche du niveau tâche, à l'exception du tcp relatif, seront connus avec la liste de leurs compétences à l'extérieur de la couche du niveau objet.

## 7.4 Un programme d'application du niveau objet

Au chapitre 6, nous avons passé en revue un ensemble de programmes d'applications du niveau "XYZ". Il figurait parmi ces applications, un programme des tours de Hanoï. Voyons ce que devient ce même programme au niveau "objet".

Hypothèses

- supposons que tous les plateaux soient caractérisés par une hauteur de 2 cm.
- Leur position de saisie se situe à mi-hauteur entre la base et le sommet du plateau sur son axe de symétrie vertical.
- La distance d'approche pour tout objet sera de 6 cm, la pente d'approche sera de 90°, la force de serrage sera relativement élevée.
- On n'utilisera pas de points intermédiaires car on a la certitude qu'il n'y aura pas de collisions, parce que la hauteur de la pince sera toujours suffisante pour permettre le passage du plateau transporté au-dessus des autres plateaux.

```
pgm :- initialisation-pince,
       définition-site-et-objet,
       hanoï.

définition-site-et-objet :-
       Approche = appr(vit(200),9000,100,5000),
       Force    = force(5,5,4),
       créer-site(piquet1,Approche,tcp(28000,0,0,9000,0,_)),
       créer-site(piquet2,Approche,tcp(21500,18000,0,9000,0,_)),
       créer-site(piquet3,Approche,tcp(4800,27500,0,9000,0,_)),
```

...

```
        créer-objet(platl,Approche,Force,ray(1000),
              tcp(28000,0,2000,9000,0,_),tcp(28000,0,1000,9000,0_)),
        créer-objet(plat2,Approche,Force,ray(2000),
              tcp(28000,0,3000,9000,0,_),tcp(28000,0,4000,9000,0,_)),
        créer-objet(plat3,Approche,Force,ray(3000),
              tcp(28000,0,5000,9000,_),tcp(28000,0,6000,9000,_)),
        créer-objet(plat4,Approche,Force, ray(4000),
              tcp(28000,0,1000,9000,0,_),tcp(28000,0,8000,9000,0,_)),
        assert(dessus(piquetl,[platl,plat2,plat3,plat4,piquetl]),
        assert(dessus(piquet2,[piquet2]),
        assert(dessus(piquet3,[piquet3]).

hanoï :- move(4,piquetl,piquet2,piquet3).

move(0,_,_,_) :-!.
move(N,A,B,C) :-
        N   0,
        M is N-l,
        move(M,A,C,B),
        déplacer-plateau(A,B),
        move(M,C,B,A),!.
déplacer-plateau(A,B) :-
        retract(dessus(A,[plateaul|Rl])),
        retract(dessus(B,[plateau2|R2]),
        déplacer(plateaul,plateau2,vit(400),tol(10000)),
        assert(dessus(A,Rl)),
        assert(dessus(B,[plateaul,plateau2,|R2])).
```

## 8. LA COUCHE DE NIVEAU "TACHE"

Jusqu'à présent, nous avons insisté sur le besoin de simplifier la programmation, d'améliorer sa portabilité et d'augmenter la puissance des primitives de programmation. La programmation de niveau "tâche" satisfait encore davantage ces besoins.

Jusqu'ici, nous avons utilisé une programmation explicite. Dans cette approche, l'utilisateur spécifie tous les mouvements que le manipulateur doit effectuer afin qu'il accomplisse une tâche donnée. Nous utilisions cette approche dans les niveaux "articulaire", "XYZ" et "objet".(*)

La programmation de niveau tâche est bien plus implicite. Elle offre à l'utilisateur un ensemble de primitives lui permettant de spécifier les actions à accomplir mais pas la manière de les accomplir pour la réalisation d'une tâche.

La spécification des actions consistera donc à donner successivement le nom d'un objet et la position dans l'espace de travail à laquelle on désire déposer cet objet.

La détermination de la manière de déplacer l'objet, c'est-à-dire le calcul d'une trajectoire de déplacement sera une fonction assurée par un planificateur de trajectoires.

La trajectoire de déplacement d'un objet est déterminée de manière à ce que aucun obstacle ne se trouve sur son parcours. Tous les objets de l'espace de travail sont des obstacles potentiels. Il est donc nécessaire de mettre à la disposition du planificateur un modèle de l'espace de travail exprimant la position courante et la géométrie de chaque objet. Sur base de ce modèle, il lui sera donc possible de trouver un chemin sans collision pour le déplacement d'un objet donné.

Dans le cas qui nous occupe, nous disposons déjà d'un tel modèle : le niveau "objet" assure la création et la mise à jour d'une représentation de l'espace de travail. Cette représentation peut être utilisée comme modèle par le planificateur car elle reprend les caractéristiques géométriques des objets, leur position,

---

(*) On remarquera que la couche "XYZ" offre déjà une programmation plus implicite que la couche articulaire. La couche "objet" offre la possibilité d'une programmation encore plus implicite.

la manière de les approcher, de les saisir, de les serrer, les relations qui les lient entre-eux ...

La couche de niveau "tâche" offre une souplesse supplémentaire à la programmation. Dans les niveaux inférieurs, on avait recours à l'utilisation de points intermédiaires (points via) pour éviter les collisions d'un objet transporté avec les obstacles. Or, l'usage de ces points intermédiaires nécessite de la part du programmeur une bonne connaissance de la position des obstacles.

L'usage des points intermédiaires n'est pas évident; d'une part, la position des obstacles varie au cours de l'exécution de la tâche car ces derniers sont aussi des objets susceptibles d'être déplacés; d'autre part, toute nouvelle redisposition des objets dans l'espace de travail nécessitera certainement la redéfinition de tous les points intermédiaires, car les obstacles ne sont plus à la même position initiale.

Le planificateur de trajectoire offre donc l'avantage de débarasser l'utilisateur du problème de la définition fastidieuse des points intermédiaires.

## 8.1 Trois catégories de planificateurs

Les techniques de planification de trajectoire peuvent apparamment être regoupées en trois catégories :

La catégorie generate and test : Les planificateurs de trajectoire les plus simples sont basés sur cette technique. Elle consiste, en toute généralité, à sélectionner des candidats et à tester pour chaque sélection la validité du candidat.

La catégorie basée sur l'usage d'une fonction de pénalité. Cette méthode consiste à attribuer un poids à des ensembles de points de l'espace et à trouver une trajectoire passant par les points de poids minimal.

La catégorie basée sur l'utilisation d'une représentation explicite de l'espace libre, l'espace qui n'est pas occupé par un objet. Les planificateurs basés sur cette technique sont de loin les plus compliqués mais semblent être les seuls qui soient réellement fiables.

Nous commencerons par proposer un planificateur basé sur la technique "generate and test". Nous parlerons ensuite brièvement des deux autres catégories.

...

### 8.1.1 Un planificateur basé sur la méthode generate and test

La méthode "generate and test" est certainement une des premières méthodes qui a été utilisée pour la planification des trajectoires. Des méthodes de planification de trajectoires basées sur la technique "generate and test" ont été développées ultérieurement par Pieper et Lewis (Pieper 1968, Lewis 1974).

La méthode que nous proposons ci-dessous est nouvelle. Elle présente plusieurs avantages. D'abord la modélisation des objets nécessaires à son fonctionnement est très simple. Ensuite elle est claire, facile à perfectionner et à implémenter. Enfin, elle offre les avantages d'une programmation du niveau tâche.

#### 8.1.1.1 La représentation des objets

Tout objet, quelles que soient sa forme géométrique, sa position et son orientation, peut être contenu dans un cylindre vertical. Ce cylindre est obtenu par la révolution de l'objet autour d'un axe de symétrie vertical perpendiculaire à la base sur laquelle il repose.

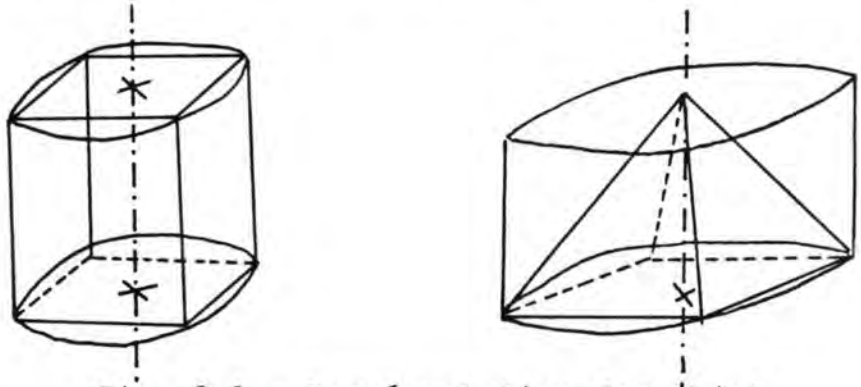


Fig. 8.1 - Représentation des objets

Avantages de cette représentation

1) La simplicité.

La représentation des objets par leur enveloppe cylindrique est aisée. Les seules informations nécessaires à cette représentation sont les suivantes :

- La position de la base permet d'identifier le point de passage de l'axe vertical de révolution;
- La hauteur de l'objet est aussi la hauteur de l'enveloppe cylindrique;

...

- Le rayon est la distance du point de l'objet qui est le plus éloigné de son axe vertical.

2) Cette représentation simplifie le raisonnement.

Les cylindres sont des volumes faciles à manipuler; de plus, ils sont tous verticaux. Cela nous permettra de raisonner sur la trace de leur projection verticale sur le plan Xo Yo de l'espace de travail. Les cylindres deviennent alors des cercles encore plus faciles à utiliser.

3) Cette représentation pallie le problème du manque de contrôle de l'orientation du solide.

Tous les robots n'offrent pas nécessairement six degrés de liberté pour les mouvements de la pince. Le robot RM-501 est dépourvu d'axe de lacet. L'angle de lacet d'un objet est donc fonction de sa position.

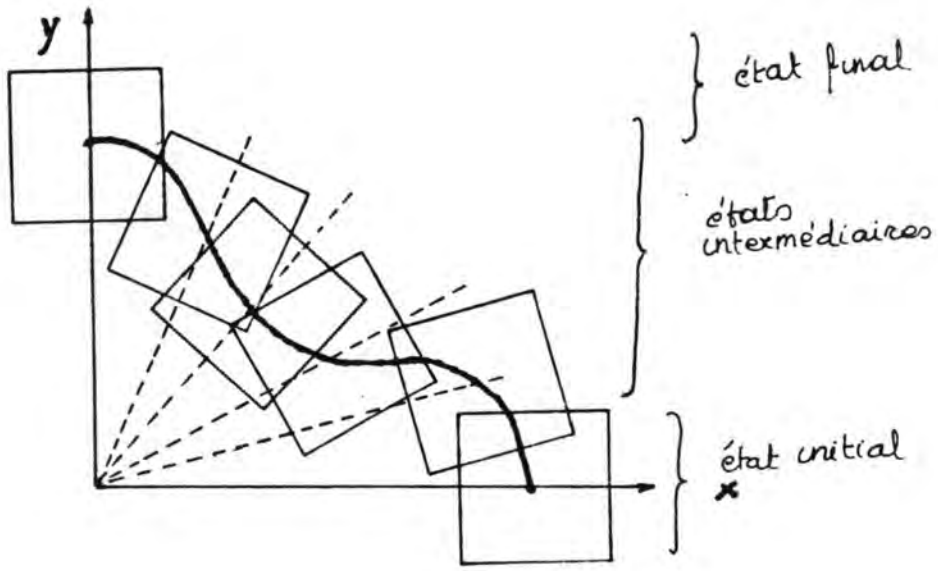
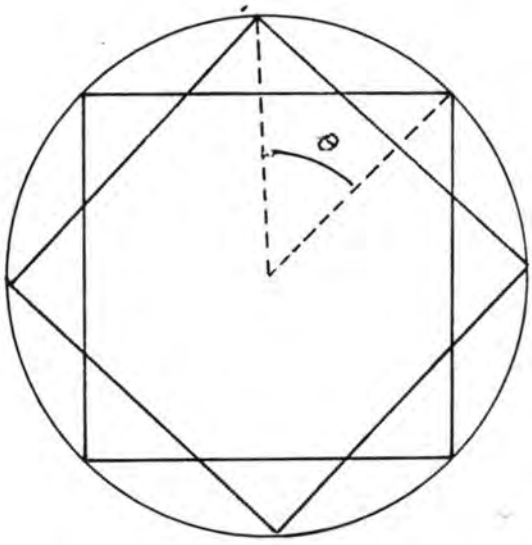



Figure 8.2. Trace du déplacement d'un objet cubique par un manipulateur dépourvu d'axe de lacet.

Nous n'avons donc pas le contrôle de l'angle du lacet. La représentation des objets par leur enveloppe cylindrique convient particulièrement pour ce cas, car un cylindre se présente toujours de la même manière quel que soit son angle de lacet.

...

Exemple




Après une rotation d'un quart de tour du solide représenté par le carré, ce dernier est toujours contenu dans le même cylindre. Si nous représentons tous les objets par des cylindres, nous n'aurons donc plus à nous soucier du problème du changement incontrôlé de leur orientation.

8.1.1.2 La détection des collisions

Pour faciliter notre raisonnement, nous utiliserons les traces circulaires des projections verticales des cylindres sur le plan horizontal contenant les axes $X_o$ $Y_o$ du repère $R_o$ de l'espace de travail. Dans un premier temps, nous allons donc ignorer la hauteur des objets. En toute généralité, lorsqu'on détecte une intersection entre deux enveloppes, cela signifie soit :

- que les deux objets enveloppés sont fort proches;
- qu'il y a une collusion entre les objets enveloppés.

Ces deux cas sont illustrés dans la figure 8.3 par la projection verticale de deux cylindres de même hauteur par rapport au plan horizontal de projection. L'intersection des enveloppes est représentée par l'intersection de leurs traces circulaires.

...

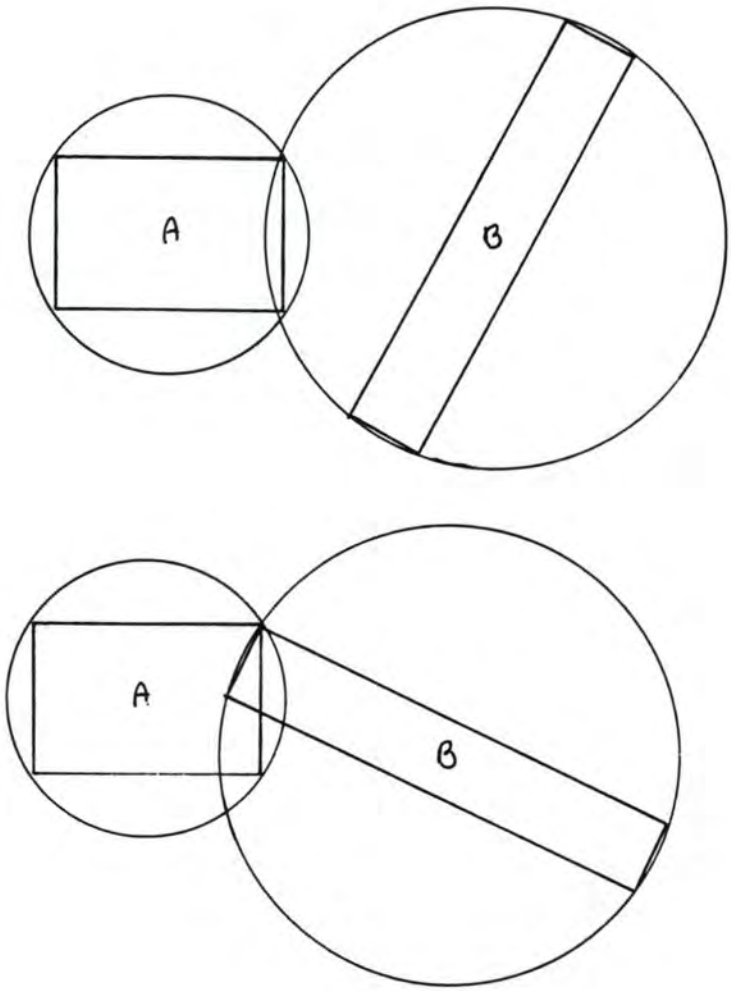



Fig. 8.3. Détection d'une collision potentielle.

D'une manière générale, si on tient compte de la hauteur des objets, il y a une collision potentielle entre deux objets :

- si la distance qui sépare le centre des 2 cercles obtenus par projection verticale sur le plan défini par les axes $X_o$ $Y_o$ du repère $R_o$ est inférieure à la somme des rayons des deux cylindres, et
- si la hauteur dans $R_o$ de tout point de la base de l'un des objets est comprise entre la hauteur de tout point du sommet de l'autre objet ou inversément.

Ce qui s'écrira d'une manière plus formalisée :

Collision(A,B) si

$$\text{rayon}(A)+(B) \geqslant \sqrt{(X(A)-X(B))^2 + (Y(A)-Y(B))2}$$

et

(hauteur-base(B)$\leqslant$ hauteur-base(A)$\leqslant$ hauteur-sommet(B)

ou

hauteur-base(A)$\leqslant$ hauteur-base(B)$\leqslant$ hauteur-sommet(A))

avec

X(K) = abscisse dans $R_o$ du point d'intersection de la base de l'enveloppe cylindrique de l'objet K avec son axe vertical de symétrie.

Y (K) = ordonnée dans $R_o$ du point d'intersection de la base de l'enveloppe cylindrique de l'objet K avec son axe vertical de symétrie.

hauteur-base(K) = hauteur dans $R_o$ du point d'intersection de la base de l'enveloppe cylindrique de l'objet K avec son axe vertical de symétrie.

hauteur-sommet(K) = hauteur dans $R_o$ du point d'intersection du sommet de l'enveloppe cylindrique de l'objet avec son axe vertical de symétrie.

8.1.1.3 <u>Sélection des obstacles situés dans le voisinage de l'objet transporté.</u>

Il est évidemment inutile de tester le risque de collision de l'objet en cours de déplacement avec tous les objets de l'espace de travail.

La plupart des objets sont suffisamment éloignés de la position courante de la base de l'objet à déplacer pour ne pas provoquer de collisions.

Les seuls candidats à la collision se situent dans un voisinage proche de l'objet en cours de déplacement. Nous pouvons sélectionner les candidats à la collision en raisonnant sur une représentation à deux dimensions des objets de l'espace de travail.

...

Soit T, l'objet à déplacer. On considère le carré de côtes parallèles aux axes $X_oY_o$ dans lequel est inscrite la projection de son enveloppe cylindrique sur le plan défini par les axes $X_oY_o$ du repère $R_o$ de l'espace de travail.

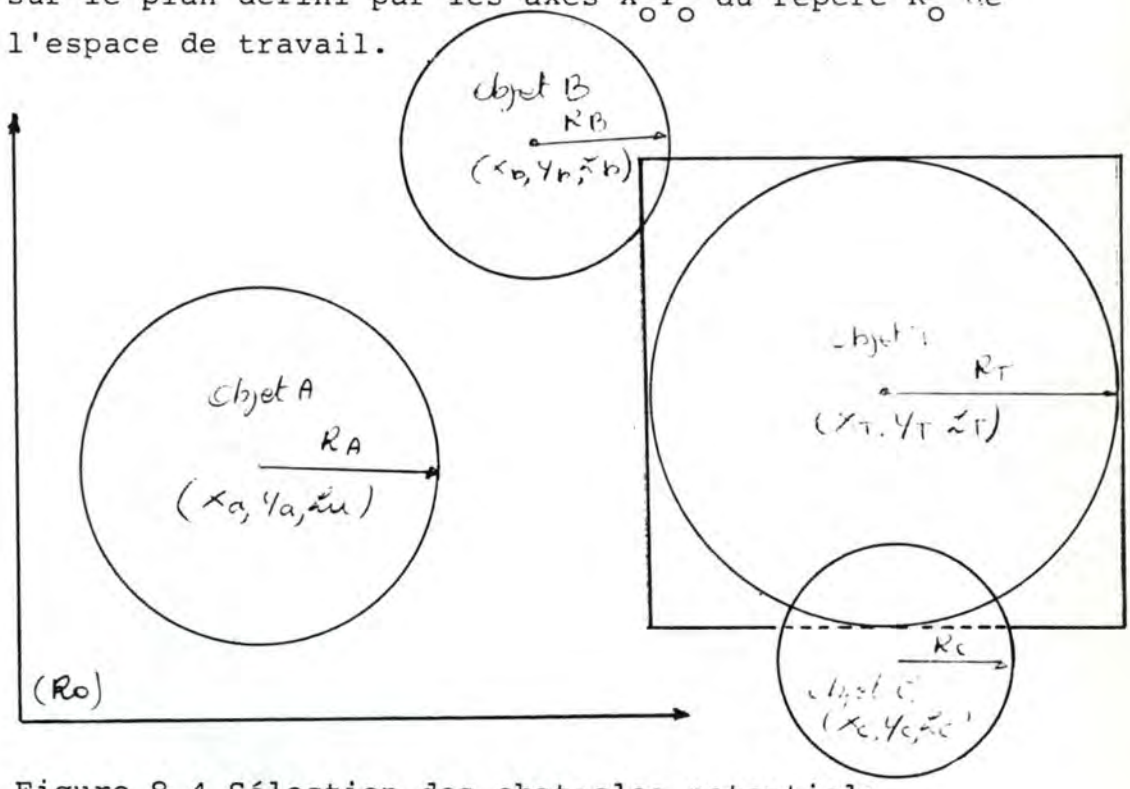



Figure 8.4 Sélection des obstacles potentiels

On retiendra comme candidats tout objet dont la projection de l'enveloppe a une intersection avec le carré dans lequel est inscrite la projection de l'enveloppe de l'objet à déplacer.
On en déduit la règle :

Candidat(D) si

$$|X_D - X_T| \leq R_D + R_T$$

et

$$|Y_D - Y_T| \leq R_D + R_T$$

Il restera alors à tester le risque de collision avec l'ensemble des obstacles sélectionnés.

8.1.1.4 Recherche d'un chemin sans collision

Nous utiliserons la méthode "generate and test" pour rechercher une trajectoire sans collision pour le déplacement d'un objet. La trajectoire obtenue sera celle que devra parcourir la base de l'objet en question.

...

Connaissant la position d'origine et de destination d'un objet, on génère une trajectoire candidate et on teste si le parcours de cette trajectoire par la base de l'objet à déplacer ne provoque pas de collisions avec d'autres objets. Si c'est le cas, la trajectoire doit être rejetée et remplacée par une trajectoire de contournement. Le même processus est alors appliqué à la nouvelle trajectoire de contournement candidate.

Les trajectoires seront constituées d'un ou de plusieurs segments rectilignes. On peut représenter facilement ces segments en ne reprenant que leurs extrémités. Ces extrémités (sauf celles du dernier segment) correspondent aux concepts de points intermédiaires exposés précedemment. Le planificateur aura donc pour rôle de dresser la liste des points intermédiaires. Dans notre approche, ces points intermédiaires permettront de constituer les tubes d'un circuit utilisés comme argument des compétences "déplacer via" définies au niveau objet.

8.1.1.5 Les trajectoires candidates

8.1.1.5.1. La trajectoire directe

La trajectoire directe est définie à l'aide d'un seul segment rectiligne. On connaît les positions initiales et finales de la trajectoire. On peut donc déterminer l'équation du segment de droite liant ces deux points.

On peut parcourir ce segment pas à pas et vérifier pour chaque position ainsi obtenue l'absence de collision avec un obstacle. Le pas peut être un paramètre laissé au choix du programmeur mais ce choix peut être un problème épineux :

- un pas trop grand pourrait provoquer le passage "au travers" d'un obstacle sans qu'il y ait eu de détection de collisions;
- un pas trop petit demandera un temps de calcul trop important.

Choissisons à tout hasard, un pas d'un demi centimètre.

...

On essayera donc de prouver que la trajectoire directe est une trajectoire sans collision.

8.1.1.5.2. La trajectoire de contournement

Lorsqu'on a prouvé la présence d'un obstacle sur un parcours d'une trajectoire, on sélectionne une trajectoire candidate de contournement de l'obstacle. Cette trajectoire passe par deux points intermédiaires afin de contourner l'obstacle qui a provoqué le rejet de la trajectoire candidate précédente.

Ensuite, on considérera le deuxième point intermédiaire comme la nouvelle position initiale d'une trajectoire qu'il reste à déterminer et qui permettra de joindre la position finale du déplacement.

- Le contournement par le dessus

Le contournement par le dessus est obtenu en calculant la position des points vial et via2.

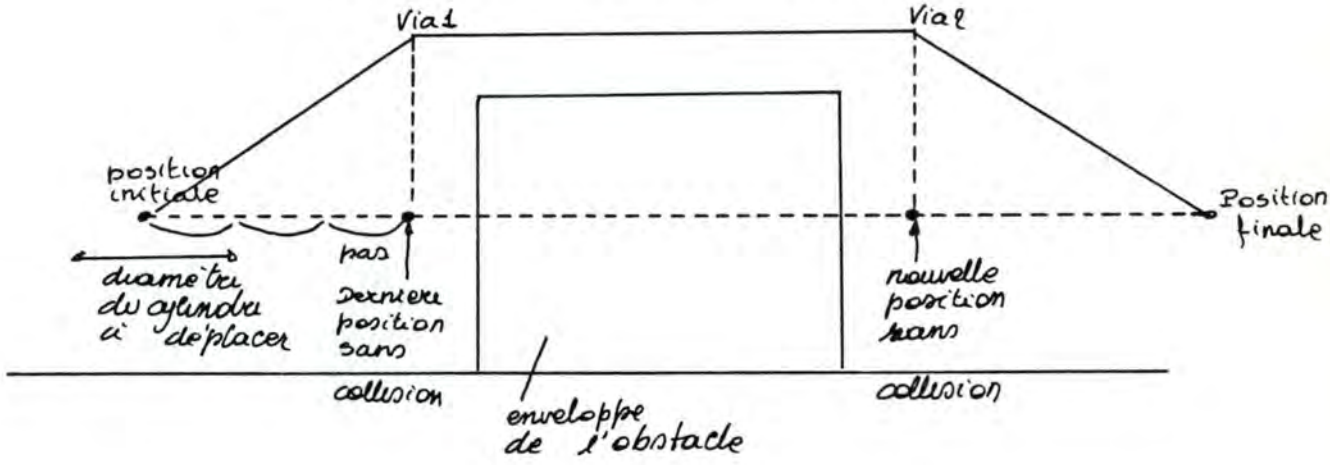



Figure 8.5 Contournement par le dessus

1° Calcul du premier point via : vial

Il suffit de retrouver la dernière position de la trajectoire initiale avec laquelle on pourrait faire coïncider la base du solide à déplacer sans qu'il y ait de collisions. Il faut alors obtenir la hauteur du sommet de l'obstacle. On obtiendra le point vial en remplaçant la hauteur de la dernière position sans collision par la hauteur du sommet de l'obstacle.

...

2° Calcul du deuxième point via : via2

On prolonge la trajectoire candidate qui a été rejetée et on la parcourt pas à pas.

Après être passé à travers l'obstacle, on obtient une nouvelle position sans collision. Le point via2 sera calculé en modifiant la hauteur de cette nouvelle position comme nous l'avons fait pour le premier point via.

La position initiale, la position finale, le point vial et le point via2 se situeront donc toujours dans le même plan vertical.

3° Détermination de la trajectoire de contournement

Il reste alors à trouver une première trajectoire liant la position d'origine au point vial, une deuxième liant les points vial et via2 et une troisième liant le point via2 au point de destination.

- Le contournement par devant et en bas

Il peut arriver que le contournement par le dessus de l'obstacle soit impossible. Il faut alors envisager de contourner l'obstacle latéralement.

Dans ce cas-ci, on évite l'obstacle en créant une trajectoire de contournement qui a tendance à rapprocher l'objet transporté de l'axe $X_o$ de l'espace de travail.

La trajectoire de contournement passe également par deux points intermédiaires obtenus par calcul. La méthode de calcul est similaire à la précédente : on retrouve la dernière position sans collision de la trajectoire candidate rejetée; on prolonge cette trajectoire et on passe à travers l'obstacle pour obtenir une nouvelle position sans collision.

...

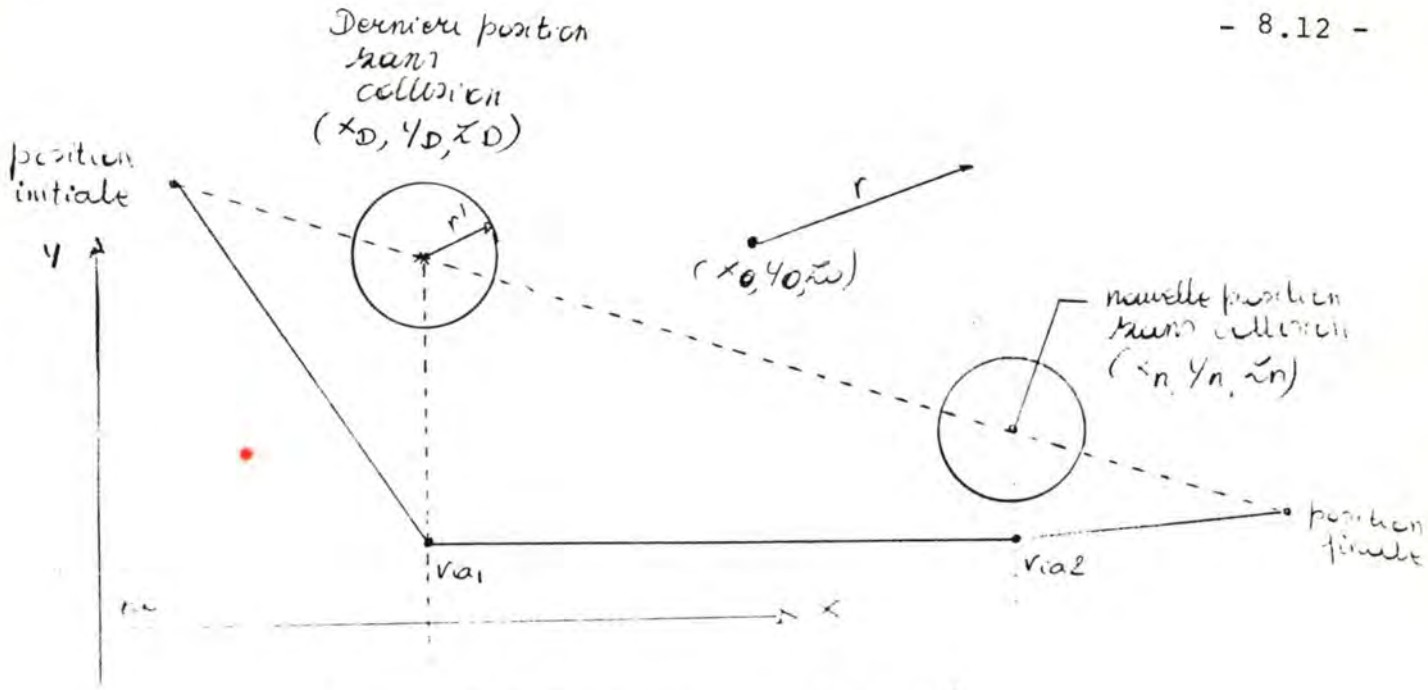



Figure 8.6 Contournement latéral

1° Calcul des coordonnées des points vial et via2

Supposons que

$X_d, Y_d, Z_d$ soient les coordonnées de la dernière position sans collision;

$X_n, Y_n, Z_n$ soient les coordonnées de la nouvelle position sans collision;

$X_o, Y_o, Z_o$ soient les coordonnées du point d'intersection de la base de l'obstacle avec son axe vertical de révolution;

r et r' les rayons respectifs de l'enveloppe de l'obstacle et de l'objet à déplacer.

Les coordonnées $X_{v1}$, $Y_{v1}$, $Z_{v1}$ du point vial seront obtenues par

$$X_{v1} = X_d$$

$$Y_{v2} = Y_o - (r + r')$$

$$Z_{v2} = Z_n$$

Les positions initiales et finales, les points intermédiares vial et via2 se situeront donc toujours sur un même plan horizontal parallèle au plan défini par les axes $X_o, Y_o$ de $R_o$.

...

2° La trajectoire de contournement

Il reste encore à trouver une première trajectoire sans collision qui lie la position initiale au premier point vial, une deuxième qui lie les points vial, via2 et une troisième qui lie le deuxième point via2 à la position initiale.

Comme pour le cas précédent, la trajectoire de contournement sera obtenue en mettant bout à bour les trois trajectoires obtenues.

- Contournement par devant et au-dessus

Le contournement se fait à la fois en rapprochant l'objet à déplacer de la base du robot et en contournant l'obstacle par le dessus. Il s'agit donc d'un cas hybride des deux précédents. La méthode est rigoureusement identique, seul le calcul des points intermédiaires est différent.
Soit $H_o$ la hauteur de l'obstacle,
on obtiendra les coordonnées du point Vial par

$$X_{v1} = X_d$$
$$Y_{v1} = Y_o - (r + r')$$
$$Z_{v1} = Z_o + H_o$$

on obtiendra les coordonnées du point via2 par

$$X_{v2} = X_n$$
$$Y_{v2} = Y_o - (r + r')$$
$$Z_{v2} = Z_o + H_o$$

- Contournement par derrière et en bas

Cas identique au cas du contournement par devant et en bas mais le contournement se fait en s'éloignant de la base du robot.

On obtiendra les coordonnées du point vial par

$$X_{v1} = X_d$$
$$Y_{v1} = Y_o + r + r'$$
$$Z_{v1} = Z_o$$

...

On obtiendra les coordonnées du point via2 par

$$X_{v2} = X_r$$
$$Y_{v2} = Y_o + r + r'$$
$$Z_{v2} = Z_o$$

- Contournement par derrière et en haut

Cas similaire au cas du contournement par devant et au-dessus. On évite l'obstacle en construisant une trajectoire de contournement qui aura tendance à éloigner l'objet de l'axe $X_o$ du repère de l'espace de travail.

On obtiendra les coordonnées du point vial par

$$X_{v1} = X_d$$
$$Y_{v1} = Y_o + r + r'$$
$$Z_{v1} = Z_o + H_o$$

On obtiendra les coordonnées du point via2 par

$$X_{v2} = X_n$$
$$Y_{v2} = Y_o + r + r'$$
$$Z_{v2} = Z_o + H_o$$

On pourrait encore imaginer d'autre scénarios de contournement mais nous nous limiterons à ceux-ci car la liste de ces scénarios risquerait d'être fort longue.

8.1.1.6. Algorithme du planificateur sous forme clausale

Dans cet algorithme, nous conviendrons d'une autre représentation de la trajectoire. Comme cette dernière est toujours constituée d'un ensemble de segments rectilignes, on peut la représenter en dressant la liste des points intermédiaires par lesquels la trajectoire devra passer, connaissant son origine et sa destination.

Si, par chance, la liste des points intermédiaires est une liste vide, cela signifiera que la trajectoire qui lie l'origine à la destination est une trajectoire directe.

...

```
trouver-un-chemin (Orig, Dest, Nom-objet, Liste-pts-interm) si
        obtenir le rayon de l'objet à déplacer,
        trajectoire (Orig, Dest, Rayon, Liste-pts-interm)

(* trajectoire rectiligne *)

trajectoire (Orig, Dest, Rayon, Liste-vide) si
        prouver qu'il n'y a pas de collisions pour tous les pas
        de la progression dans la trajectoire rectiligne joignant
        le point Orig au point Dest.

(* contournement par le haut *)

trajectoire (Orig, Dest, Rayon, Liste-pts-interm) si
        retrouver les caractéristiques géométriques de l'obstacle,
        retrouver la dernière position sans collision,
        trouver la nouvelle position sans collision,
        trouver le point vial permettant de contourner l'obstacle
             par le dessus,
        trouver le point via2 permettant de contourner l'obstacle
             par le dessus,
        trajectoire (Orig, vial, Pv1),
        trajectoire (vial, via2, Pv2),
        trajectoire (via2, Dest, Pv3),
        concatener (Pv1, [vial], Pv2, [via2], Pv3) et affecter
             le résultat à Liste-pts-interm.

(* contournement par devant et en bas *)

trajectoire (Orig, Dest, Rayon, Liste-pts-interm) si
        retrouver les caractéristiques géométriques de l'obstacle,
        retrouver la dernière position sans collision,
        trouver la nouvelle position sans collision,
        trouver le point vial permettant de contourner l'obstacle
             par devant et en bas,
        trouver le point via2 permettant de contourner l'obstacle
             par devant et en bas,
        trajectoire (Orig, vial, Pv1),
        trajectoire (vial, via2, Pv2),
        trajectoire (via2, Dest, Pv3),
        concatener (Pv1, [vial], Pv2, [via2], Pv3) et affecter
             le résultat à Liste-pts-interm.
```

Pour le contournement par devant et en haut, le contournement par derrière et en bas et le contournement par derrière et en haut la méthode est similaire à la précédente.

...

8.1.1.7. Les améliorations

La méthode de base peut être améliorée.

- la trajectoire déterminée par le planificateur est celle que devra parcourir la base de l'objet à déplacer. Elle n'est donc pas nécessairement égale à la trajectoire de la pince (en toute généralité, ces deux trajectoires sont parallèles).
  Il faut veiller à ce que toutes les positions de cette dernière trajectoire soient compatibles avec les contraintes mécaniques et géométriques du manipulateur qui porte la pince car celle-ci ne peut en aucun cas sortir de l'espace de travail. En cas d'incompatibilité, la trajectoire candidate est à rejeter. On peut limiter le test aux points "intermédiaires" qui sont des points critiques. Nous devons pouvoir utiliser les primitives de déplacement définies dans la couche du niveau objet. L'ensemble des points intermédiaires calculés par le planificateur permettra de construire un ensemble de tubes. On construira un circuit à l'aide des tubes qui servira d'argument à la primitive "déplacer via" du niveau objet. Pour constuire un tube, il faut encore une tolérance et celle-ci doit être aussi grande que possible. On pourrait envelopper l'objet à déplacer d'un deuxième cylindre de rayon supérieur à celui de l'enveloppe de l'objet.
  Pour chaque segment rectiligne de la trajectoire sans collision,on essayera de trouver un rayon maximum pour cette deuxième enveloppe, mais suffisamment restreinte pour que cette dernière n'entre en collision avec aucune autre enveloppe. La différence entre les deux rayons représentera la tolérance recherchée
- il reste encore à choisir une vitesse pour ces tubes. On pourrait prendre pour vitesse une vitesse identique à celle donnée comme argument à la fonction "déplacer-via".

...

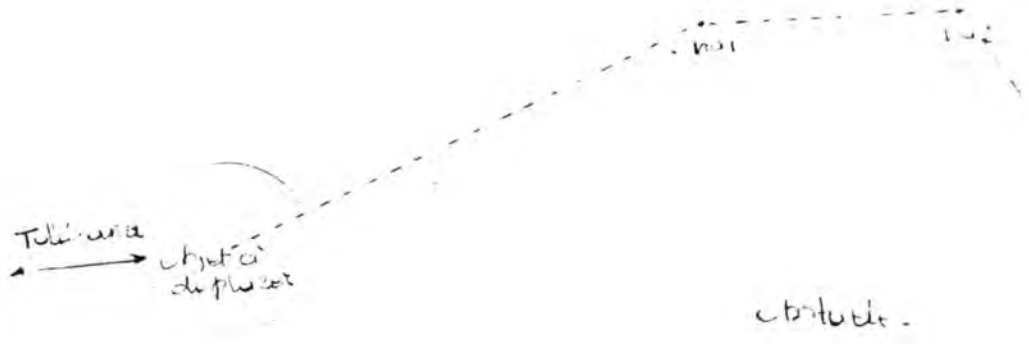


Figure 8.7 Détermination d'une tolérance pour la trajectoire

- dans la couche du niveau objet, nous avons émis l'hypothèse que certains objets pouvaient être liés. Si deux objets sont liés, le déplacement de l'un provoque le déplacement simultané de l'objet est solidaire
  Il faudra donc trouver un chemin tel que son parcours par la base d'un objet ne provoque pas de collisions. Les objets solidaires transportés simultanément selon une trajectoire parallèle ne peuvent également pas provoquer des collisions.
  La pince est aussi un objet susceptible de provoquer des collisions. Lorsqu'on déplace un objet à l'aide de la pince, il conviendrait alors de le lier à celle ci pour déterminer une trajectoire sans collision à la fois pour la pince et l'objet déplacés.

8.1.1.8. Limites de la méthode

Cette méthode fonctionne théoriquement bien mais peut être sujette à certaines défaillances.

- la méthode détecte des collisions alors qu'il n'y en a pas. Cette erreur est due à la simplicité de la représentation des objets. Cette erreur, en soi, n'est pas très grave mais elle peut en provoquer de bien plus graves car, le planificateur peut arriver à la conclusion qu'il n'y a pas de trajectoire sans collision possible alors qu'en réalité, il y a moyen d'en trouver une.
- la méthode provoque l'évitement de justesse de l'objet

...

transporté avec les obstacles. En effet, l'objet transporté rase toujours les obstacles. La trajectoire qui en résulte est souvent "zigzaguante". D'autres méthodes permettent d'éviter "généreusement" les obstacles. La trajectoire en résultant est donc bien plus naturelle et souple.

- la méthode retient le premier chemin sans collision mais pas le meilleur (le plus court ou le plus rapide, par exemple).
- la méthode peut aussi échouer lorsque l'espace de travail est particulièrement encombré. Dans ce cas, le contournement d'un obstacle peut typiquement provoquer la collision avec un autre et ainsi de suite. Le danger de collision en chaîne est encore plus grand lorsque la pince transporte des objets liés.
- la méthode recherche un chemin dont le parcours par la base d'un objet ne provoque pas de collision avec les obstacles. Si l'objet déplacé ne provoque pas de collision, il n'en est pas nécessairement de même pour la pince, l'avant-bras, le coude, l'arrière-bras et le corps du robot. On peut toutefois tenir compte de la présence de la pince en la considérant également comme un objet à transporter (cf améliorations). Cette amélioration ne tient pas encore compte de la présence des autres membres et il subsiste donc toujours un risque de collision. On peut se permettre de le courir, car il n'est pas aussi important. Déjà au niveau "XYZ", nous ignorions la présence du porteur de la pince et de la pince elle-même en ne considérant que sont extrémité. D'autre part, nous pouvons constater que le bras du robot RM 501 surplombe toujours l'espace de travail car le coude est toujours situé vers le haut. Enfin, le RM 501 est un micro-robot; il ne permet donc que la manipulation des petits objets, ce qui réduit encore le risque de collisions imprévues.

...

8.1.1.9. Recommandations pour la disposition des objets dans l'espace de travail

Nous avons vu que le planificateur ne fonctionnait pas toujours. Il y a moyen d'éviter des situations insolubles en disposant les objets d'une certaine manière.

Il faut d'abord éviter une trop grande proximité des objets pour permettre le passage d'une trajectoire de contournement.

Il faut essayer de placer les obstacles particulièrement hauts et encombrants le plus loin possible de la base du robot.

Il faut éviter de placer les objets trop près des limites de l'espace accessible par le bras du robot.

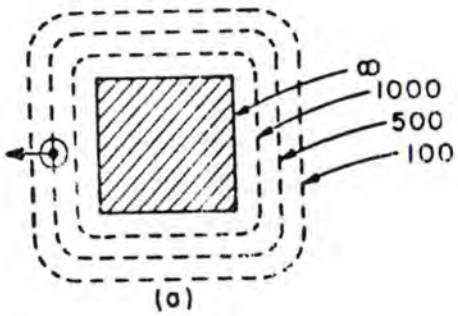
Il faut bien sûr, éviter d'encombrer excessivement l'espace de travail par la présence d'un trop grand nombre d'objets.

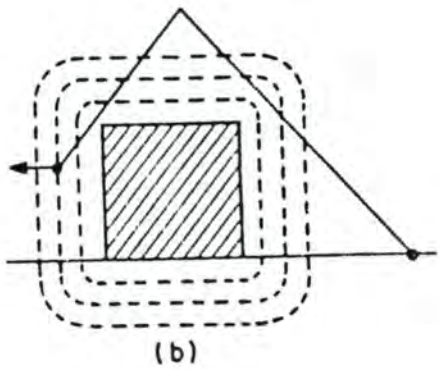
8.1.2. L'utilisation de la méthode des pénalités

(Khatib 1980, Hogan 1981)

La méthode des pénalités est attirante parce qu'elle offre un moyen simple d'exprimer la combinaison de contraintes imposées par les objets de l'espace de travail.

Les objets sont entourés d'une série d'enveloppes concentriques. On attribue à l'enveloppe la plus proche une pénalité infinie et on attribue des pénalités décroissantes aux autres enveloppes, à mesure qu'elles s'écartent de l'objet entouré. Le planificateur devra trouver un chemin catactérisé par la pénalité la plus faible.




...

### 8.1.3. La méthode utilisant une représentation explicite de l'espace libre

(Lozano-Perez - Brooks, 1981)

Cette méthode a été développée par Lozano-Perez et Brooks. L'espace libre est représenté explicitement en exagérant le volume occupé par les obstacles. Cette exagération est fonction de la géométrie et des dimensions du volume à déplacer. Tout objet est représenté par la réunion de polyhèdres convexes. Le nombre de polyhèdres, le nombre de leur surface est un problème non trivial. On peut représenter les objets très fidèlement mais il ne faut pas pousser à l'excès la précision du modèle. Un modèle complexe nécessite beaucoup de temps de calcul pour son utilisation.

#### 8.1.3.1. La théorie du "growing obstacle space"

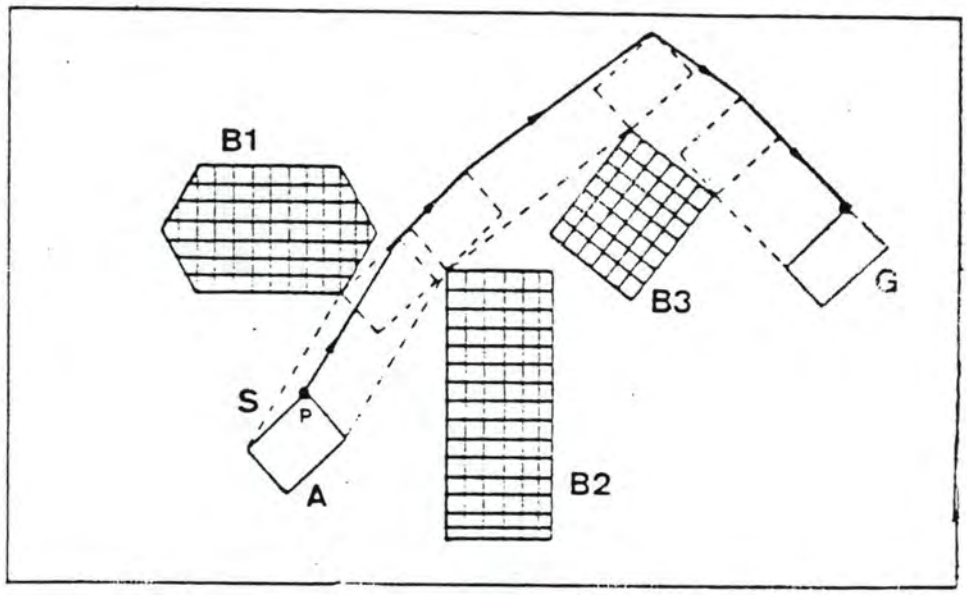
Cette théorie constitue le coeur de cette méthode de génération de trajectoire sans collision. Le principe est le suivant :

- on désigne un point p de l'objet à déplacer. La trajectoire générée sera celle que devra suivre ce point. Ce point pourrait correspondre à la position de la base ou encore à la position de saisie du solide à déplacer.
- le volume de tous les obstacles est exagéré (growing obstacles) dans certaines proportions qui sont fonction de l'objet à déplacer. Toute position de l'espace de travail qui ne se situe pas dans l'un de ces volumes, est une position sans collision par laquelle la trajectoire du point p peut passer.
  Comme l'espace libre pour le déplacement de chaque objet est représenté explicitement, on peut alors déterminer un chemin optimal de déplacement sans collision. Ce chemin pourra, par exemple, éviter les obstacles d'une manière plus "généreuse" qu'auparavant

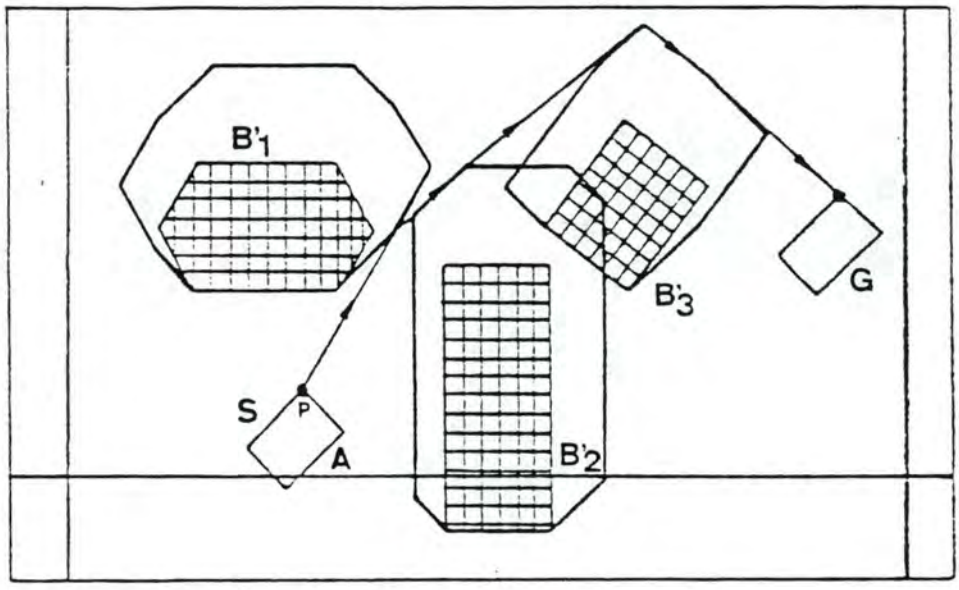
Pour expliquer la méthode de Lozano-Perez, nous allons nous limiter à un espace à deux dimensions. Nous allons traiter simultanément le problème de la recherche de l'espace libre (findspace) et celui de la recherche d'une trajectoire dans cet espace libre (findpath).

...

8.1.3.2. Recherche d'un chemin sans collision pour le déplacement d'un objet sans modification de son orientation




On demande de déplacer l'objet A de la position S à la position G. L'orientation de l'objet A doit être identique au cours de son déplacement.




...

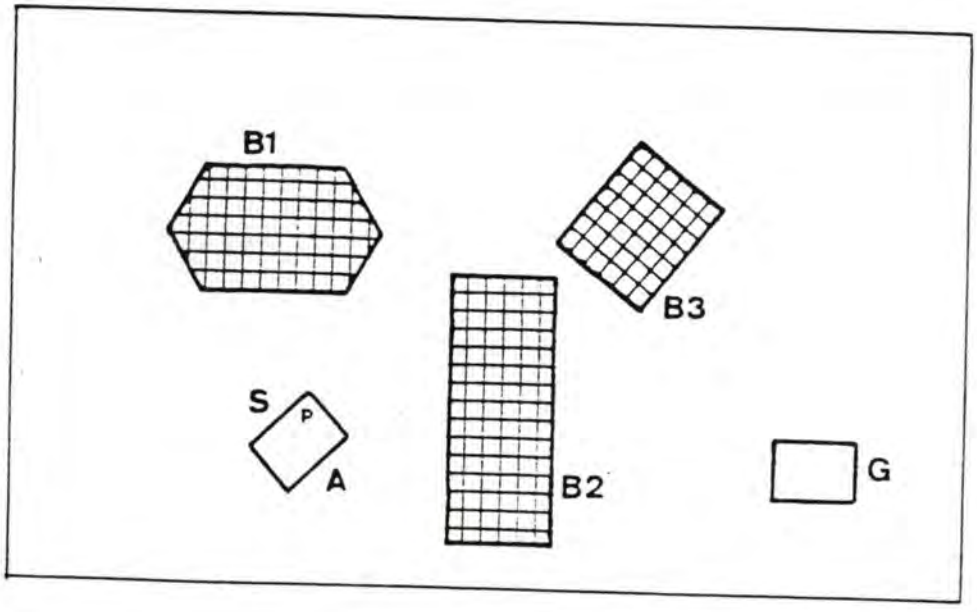
Dans la figure 8. , on remarquera que le volume de tout obstacle $B_i$ a été augmenté pour obtenir $B'_i$.

Le point P est le point de référence pour lequel il faudra rechercher une trajectoire. Lorsque P se situe à l'extérieur de tous les volumes $B'_i$, l'objet A ne rentre pas en collision avec les obstacles $B_i$. On peut vérifier cette propriété visuellement sur la figure 8. .

Il reste encore à rechercher les points intermédiaires par lesquels la trajectoire devra passer. Le chemin le plus court de S à G passera par les arrêtes (ici les coins) des polyhèdres $B'_i$. Les points intermédiaires de la trajectoire la plus courte se situeront donc toujours sur ces arrêtes.

8.1.3.3. Recherche d'un chemin sans collision pour le déplacement d'un objet avec modification de son orientation

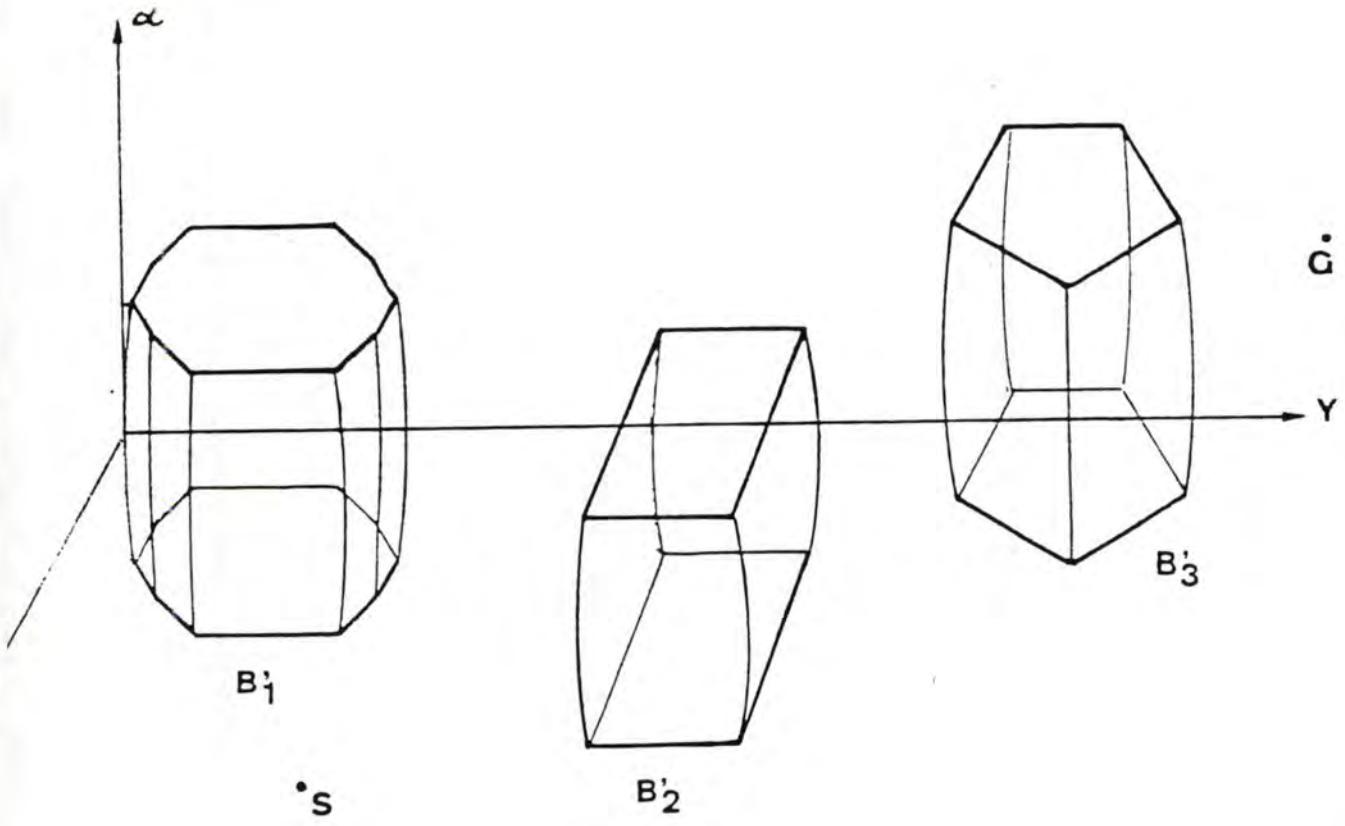
Dans le paragraphe 8.1.3.2., nous avions en fait convenu que l'objet ne pouvait se déplacer que selon deux degrés de liberté (translation en X et Y). On peut doter l'objet d'un degré de liberté supplémentaire en rotation. Cette fois-ci, nous permettons le changement d'orientation de ce volume. Le problème se pose donc dans un espace $(X,Y,\alpha)$. Les dimensions des polyhèdres $B'_i$ sont fonction de la géométrie des dimensions et de l'orientation courante du solide à déplacer.




...

Dans la figure 8. , nous constation que l'orientation des positions S et G sont différentes. La recherche d'un chemin optimal est devenue un peu plus compliquée qu'auparavant car la taille des volumes $B'_i$ n'est plus fixe.

Le problème posé dans la réalité est encore plus complexe : l'espace de travail du robot est de dimension 3 et tout objet peut être doté théoriquement de six degrés de liberté, dont trois agissent sur son orientation. Le problème se pose alors dans un espace $(X,Y,Z,\alpha,\beta,\gamma)$.




...

# 9. CONCLUSIONS

Il n'y a rien de plus évident pour un être humain que d'avoir à saisir un objet déposé sur une table encombrée et de le déposer ailleurs sur cette même table.

La même opération effectuée à l'aide d'un robot est par contre très peu evidente. L'utilisation d'un manipulateur nécessite une décomposition du mouvement. On doit d'abord amener la pince à proximité de l'objet. Ensuite, on entame une phase d'approche de l'objet pendant laquelle la pince parcourt une trajectoire extrêmement contrôlée. Lorsque la pince est enfin située sur la position de saisie de l'objet, on la serre avec une force qui ne peut être excessive. Il reste encore à soulever l'objet selon une trajectoire fortement contrôlée et enfin, on peut entamer la phase de déplacement proprement dite. Nous sommes déjà loin du mouvement identique qu'un être humain pourrait accomplir avec nonchalance.

La couche "XYZ" offre un outil qui permet de décrire un tel mouvement en permettant de contrôler chacune de ses phases. La couche "XYZ" simplifiait déjà considérablement la tâche de programmation et cette tâche s'avère encore fort complexe. C'est dire à quel point la programmation au niveau de l'actionneur est impossible. Nous avons ensuite voulu cacher toutes ces phases et simplifier autant que possible la programmation des déplacements aussi élémentaires. Nous avons alors défini une base de données dans laquelle on reprenait les caractéristiques des objets, comme leur position, la manière de les saisir, de les approcher, de les déposer. Les primitives de déplacement qui utilisent cette base de données permettent d'effacer toutes traces explicites des phases d'approche, d'élévation, de serrage et d'éloignement. Par contre, la phase de déplacement proprement dite est encore bien explicite. Ce déplacement s'avère encore fort compliqué : lorsqu'un être humain, même maladroit, déplace un objet, il va de soi que ce déplacement se fait sans collision avec les autres objets présents. Pour un robot, cela ne va pas de soi ! Il faut encore lui préciser explicitement les points par lesquels la trajectoire de déplacement devra obligatoirement passer afin d'éviter les collisions. Pour tout déplacement au niveau de l'objet, nous devons encore définir explicitement les trajectoires de contournement. On précise une

...

# INDEX DES FIGURES



...

...

# REFERENCES


Références du chapitre 1

- "La Robotique" - Opérations Athéna (farde de documentation)
  Melchior Wathelet : ministre des technologies nouvelles.
- Robot programming - Tomas Lozano-Perez
  Proceedings of IEEE vol. 71 n° 7 - July 1983
  Robot programming
- Bulletin mensuel des technologies nouvelles n° 1
  Avril 1984 - "La Robotique"

Références du chapitre 2

- Zomer cursus : mathematica, programmatie en controle van industriele robots - 4, 5 Juli 1983 - J. SIMONS
  Faculteit der toegepaste wetenschappen K.U.L.
- Les Robots - tome 5 - Michel Parent
  Claude Lourgeau 1983
  Langages et méthodes de programmation - éditions Hermès
- Introduction à la robotique vol. 1 - Pierre Lopez
  Jean-Numa Foulc 1984
  éditions éditest

Références du chapitre 5

- Les Robots - tome 1 - Philippe Coiffet
  Modélisation et commande
  Editions Hermès 1981
- Robot manipulators - Richard P. Paul
  Mathematics, Programming and control - 1981 - MIT
- Introduction à la robotique vol. 1 - Pierre Lopez
  Jean-Numa Foulc 1984

Références du chapitre 8

- Robot Motion : Planning and control - Michael Brady
  - John Hollerbach
  - Timothy Johnson
  - Tomas Lozano-Perez
  - Mattew-Mason

  (MIT)

- Lozano-Perez, T. Automatic planning of manipulator transfer movements. IEEE Trans. Systems Man Cybernet 11 (1981) 681-698
- Brooks R.A., Solving the findpath problem by good representation of free space. IEEE Trans. Systems Man Cybernet 13 (1983) 190-197
- Brooks R.A. and Lozano-Perez, T. A subdivision algorithm in configuration space for findpath with rotation in : Proceedings Eighth International Joint Conference on Artificial Intelligence. Karlsruhe, W. Germany (1983)
- Zomer cursus : mathématica, programmatie en controle van industriele robots - 4, 5 Juli 1983 - J. SIMONS Faculteit der toegepaste wetenschappen K.U.L.

# ANNEXE A

## PROGRAMMATION DU NIVEAU ARTICULAIRE

# ANNEXE A : PROGRAMMATION DU NIVEAU ARTICULAIRE

## LES COMPENTENCES DU ROBOT PHYSIQUE

Le constructeur a défini un jeu de 32 commandes pour programmer le robot. Ces commandes sont interprétées par un interpréteur câblé. Dans le cas qui nous occupe, ces commandes ne seront pas toutes utiles, nous n'utiliserons qu'un sous-ensemble de neuf commandes dont les caractéristiques sont énoncées ci-dessous.

### LA COMMANDE NEST

La commande "Nest"a pour effet de réinitialiser le robot dans une position connue de l'unité de commande. Chaque membre subit une rotation jusqu'à ce que l'interrupteur de fin de course soit ouvert.

Le contrôleur provoque tour à tour la rotation des axes de chaque corps jusqu'à ouverture du contact de fin de course.

La séquence des mouvements est la suivante :

- rotation de l'épaule vers le haut;
- rotation du coude vers le bas;
- rotation du poignet vers le bas et selon son axe de tangage;
- rotation du poignet dans le sens anti-horlogique selon son axe de roulis;
- rotation du corps dans le sens anti-horlogique.

Lorsque le mouvement est terminé, les membres du robot se trouvent dans une position que l'on appelle l'origine mécanique. L'origine mécanique représente aussi la home position par défaut.

Format de la commande : "NT"↵

### LA COMMANDE DE POSITION' SET

L'exécution de la commande a pour effet de mémoriser une configuration des membres du robot dans la mémoire de son unité de commande. Cette configuration est exprimée à l'aide d'un nombre de pas représentant la rotation de chaque articulation par rapport à la home position.

...

Format de commande

PS $a_o$, a1,a2,a3,a4,a5,a6↵

Les paramètres $a_i$ sont des nombres entiers.

Toutes les positions mémorisées dans la mémoire de l'unité de commande du robot portent un numéro de position . Ce numéro de position est exprimé par le paramètre $a_o$ et servira à identifier la position mémorisée entre 0 unité et 629.

Les paramètres a1 à a5 expriment en nombre de pas l'angle de rotation de chaque articulation par rapport à une position de référence; ce sont donc des variables articulaires.

Les tableaux T1 et T2 ci-dessous expliquent plus en détail la signification des paramètres et la manière de les utiliser.

| PARAMETRE | ARTICULATION | SIGNE + | SIGNE - | AMPLITUDE D'UN PAS |
|---|---|---|---|---|
| $a_1$ | Pivot du corps | sens ↻ | sens ↺ | 0,025 ° |
| $a_2$ | épaule | sens ↑ | sens ↓ | 0,025 ° |
| $a_3$ | coude | sens ↑ | sens ↓ | 0,025 ° |
| $a_6$ | - | - | - | - |

Le paramètre a6 représente le lacet de l'organe terminal mais il n'y a pas d'axe de lacet, ce paramètre doit toujours avoir une valeur nulle.

Paramètres a4 et a5 : ces deux paramètres indiquent l'orientation de l'organe terminal du robot. Le tableau T2 exprime le sens du mouvement de l'organe terminal pour une même valeur absolue des paramètres a4 et a5.

| signe $a_4$ | signe $a_5$ | mouvement du poignet |
|---|---|---|
| + | + | roulis : sens ↻ |
| - | - | roulis : sens ↺ |
| + | - | tangage : sens ↑ |
| - | + | tangage : sens ↓ |

...

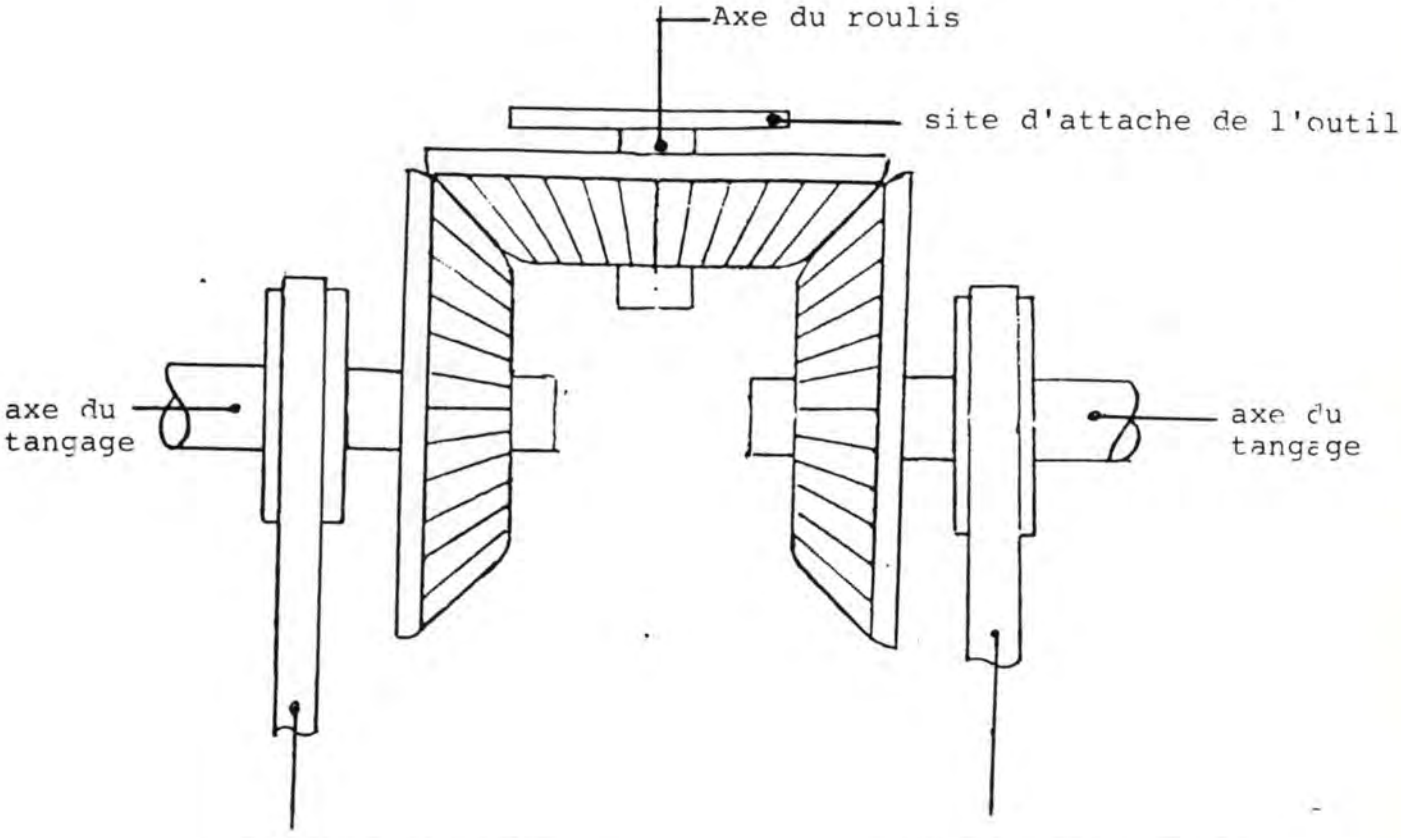
## UTILISATION DES PARAMETRES A4 ET A5 POUR L'ORIENTATION DE L'ORGANE TERMINAL

L'orientation de l'organe terminal est assurée par un mécanisme à différentiel. Ce mécanisme est avantageux parce que l'axe de tangage et de roulis sont concourants. La rotation des engrenages du différentiel est assurée par le mouvement de deux courroies fixées sur des poulies. Ces courroies sont mises en mouvement par deux actionneurs indépendants dont le mouvement est fonction respectivement des paramètres a4 et a5. Ces paramètres expriment un nombre de pas. Chaque pas est un angle de 0,075°. Un mouvement d'inclinaison du poignet par rotation de l'axe de tangage sera obtenu par addition des mouvements de rotation des deux poulies.

Si les deux poulies tournent "dans le même sens", le signe des paramètres a4 et a5 est opposé, car le moteur des actionneurs assurant le mouvement des courroies tourne en sens opposé à cause de la symétrie de l'architecture du robot. Dès lors, si on désire effectuer un mouvement de tangage de m pas, il faudra que les deux poulies effectuent un mouvement de rotation de m pas "dans le même sens"; donc la valeur des paramètres a4 et a5 sera respectivement +m et -m. Si on désire effectuer un mouvement de roulis, de n pas, il faudra faire tourner les deux poulies "dans un sens opposé". Le signe des paramètres a4 et a5 sera donc identique et leur valeur représentera un nombre n de pas.

De manière générale, si on désire effectuer un mouvement de tangage de m pas et un mouvement de roulis de n pas alors la valeur des paramètres a4 et a5 sera :

$$a4 = n + m$$
$$a5 = n - m$$




LA COMMANDE "MOVE"

Format de la commande "MO a" ↵

$0 \leq a \leq 629$

La commande "move" provoque une rotation simultanée des cinq articulations du robot et amène les corps qui le composent dans une position identique à la position portant le numéro a. Si cette position n'existe pas, le robot se met en mode erreur (lampe rouge allumée sur la face frontale de l'unité de commande).

LA COMMANDE "MOVE CONTINOUSLY"

Format de la commande "MC a" ↵

$-629 \leq a \leq 629$

L'exécution de cette commande provoque le passage sans interruption du mouvement par "a" configurations successives.

Ces configurations sont mémorisées au préalable à l'aide de la commande "position set".

...

Si "a" est positif, ces positions sont parcourues par ordre croissant de numéro de position. "Si "a" est négatif, elles sont parcourues par ordre décroissant.

L'exécution de la commande "MC" doit être précédée de l'exécution de la commande "move". Cette dernière servira à préciser le numéro de la position de départ du parcours continu.

La valeur absolue de "a" doit être inférieure ou égale au nombre de positions mémorisées antérieurement à l'aide de la commande de "position set". Si ce n'est pas le cas, le robot se met en mode "erreur".

Remarque : Une précaution s'impose lors de l'utilisation de la commande "MC". Il faut éviter d'inverser le sens de rotation des moteurs lorsque ceux-ci tournent à haute vitesse. Ces inversions brusques peuvent détériorer le mécanisme du bras articulé.

Exemple : Supposons que l'on ait enregistré les positions n°1, 4, 5, 10, 50, le parcours de ces positions par numéro croissant de numéro de position s'écrira de la manière suivante :

```
MO 1
MC 4
```

## LA COMMANDE "POSITION CLEAR"

Format de la commande PC $a_1$, [$a_2$] ↵

$a_1$ $a_2$

$1 \leq a_1, a_2 \leq 629$

La commande "clear position" a pour effet d'effacer toutes les positions dont le numéro de position appratient à l'intervalle $a_1$ ..... $a_2$.

Le paramètre $a_2$ peut être omis; dans ce cas, toutes les positions dont le numéro de position est supérieur ou égal à la valeur du paramètre $a_1$ seront effacées.

## LA COMMANDE "GRIP OPEN"

Format de la commande GO ↵

L'exécution de la commande "go" provoque une ouverture de la pince de l'organe temrinal du robot.

...

LA COMMANDE "SET GRIP PRESSURE"
==========

Format de la commande GP $a_1$, $a_2$, $a_3$ ↵

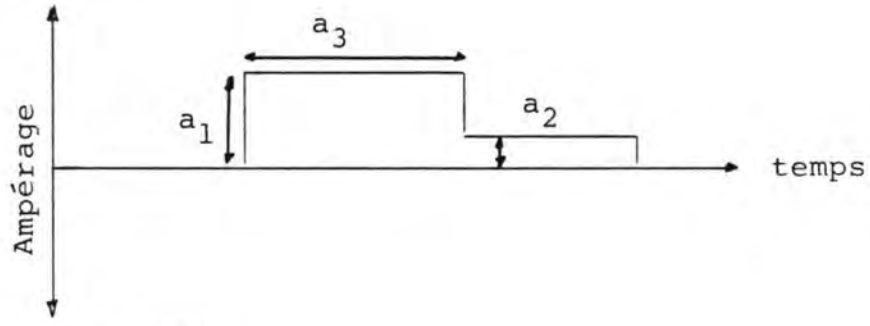
$0 \leq a_1, a_2 \leq 7$

$0 \leq a_3 \leq 99$

Cette commande contrôle la tension d'alimentation du moteur de la pince. La pression de la pince est proportionnelle à cette tension.

Les paramètres $a_1$ et $a_2$ désignent une pression dont la valeur varie entre 0 et 7.

Le paramètre $a_3$ désigne la durée, exprimée en dixièmes de secondes, pendant laquelle la pince exerce une pression exprimée par le paramètre $a_1$.




LA COMMANDE "SPEED SET"
==========

Format de la commande SP a ↵

0 a 9

La commande "speed set" représente la vitesse maximale de tout nouveau mouvement.

La vitesse est représentée par le paramètre a.

La vitesse minimale est de 40 mm/seconde. Elle correspond à la valeur a = 0.

La vitesse maximale est de 400 mm/seconde et correspond à la valeur a = 9.

Les vitesses intermédiaires sont approximativement proportionnelles à la valeur de a.

...

LA COMMANDE "GRIP CLOSE"
==========

Format de la commande GC ↵

L'exécution de la commande "grip close" provoque une fermeture de la pince avec une force déterminée par la commande "grip pressure set".

LA COMMANDE "GRIP FLUG"
==========

Format de la commande GF a↵
a = 0 ou 1

Cette commande associe l'état de la pince lors de la mémorisation de position.

Lorsque GF 1 est exécuté, on associera l'état "pince fermée" de la pince à toute nouvelle position enregistrée.

Lorsque GF 0 est exécuté, on associera l'état "pince ouverte" de la pince à toute nouvelle position enregistrée.

Exemple : GF 1
PS 10, 10, 20, 30, 40, 50, 0
MO 10
provoquera le positionnnement du robot dans la configuration représentée par la position numéro 10 ainsi que la fermeture de la pince.

LA COMMANDE "TIME SET"
==========

Format de la commande $T_i$ a ↵
0 a 99

L'exécution de la commande "time set" provoque un arrêt de toute activité du robot pendant un temps a x 0,1 seconde.

...

EXEMPLE DE PROGRAMME (Mode immédiat)

Le programme ci-dessous représente, même si ce n'est pas très évident, le premier mouvement du bras du robot permettant de sésoudre le problème des tours de HANOI.

```
NT
PS 0,0,-4100,3600,1450,-950,0
SP 9
PS 1,0,4100,-3600,-1450,950,0
PS 2,-3204,1823,-3364,-686,686,0
MO 1
MC 1
PC 1
PS 1,-3204,1823,-3364,-686,686,0
PS 2,-3204,1748,-3401,-649,649,0
PS 3,-3204,1670,-3431,-613,613,0
PS 4,-3204,1589,-3452,-578,578,0
PS 5,-3204,1504,-3466,-545,545,0
PS 6,-3204,1415,-3472,-514,514,0
MO 1
MC 5
PC 1
PS 1,-3204,1415,-3472,-514,514,0
PS 2,-3204,1324,-3469,-484,484,0
PS 3,-3204,1230,-3459,-456,456,0
MO 1
MC 2
PC 1
GP 7,5,5
GC
GF 1
PS 1,-3204,1230,-3459,-456,456,0
PS 2,-3204,1823,-3364,-686,686,0
MO 1
MC 1
PC 1
PS 1,-3204,1823,-3364,-686,686,0
PS 2,0,1816,-3348,-689,689,0
MO 1
MC 1
PC 1
PS 1,0,1816,-3348,-689,689,0
PS 2,0,1742,-3386,-652,652,0
PS 3,0,1665,-3415,-616,616,0
PS 4,0,1584,-3437,-582,582,0
PS 5,0,1499,-3450,-549,549,0
PS 6,0,1411,-3456,-518,518,0
MO 1
MC 5
PC 1
PS 1,0,1411,-3456,-518,518,0
PS 2,0,1320,-3454,-488,488,0
PS 3,0,1227,-3444,-460,460,0
PS 4,0,1131,-3426,-435,435,0
PS 5,0,1033,-3400,-411,411,0
MO 1
MC 4
PC 1
GO
GF 0
```

# ANNEXE B

## IMPLEMENTATION EN PROLOG DE LA COUCHE DE NIVEAU ARTICULAIRE

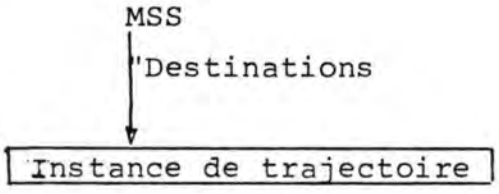
Dans le chapitre 4, nous avons abordé une première modélisation de la couche articulaire par le biais d'une approche orientée objet.

Dans cette annexe, nous allons nous pencher sur l'implémentation des objets et de leurs compétences.

Dans l'approche orientée objet, nous avons des objets caractérisés par une série de compétences.

On active une compétence en envoyant un message identifiant cette compétence. La cible du message sera une instance d'objet.

Nous aurons par exemple :




ce qui aura pour effet de donner une position représentant la destination de la trajectoire. Pour l'implémentation en Prolog de la couche, nous utiliserons des prédicats dont le foncteur sera le nom d'une compétence. On aura par exemple

destination(Trajectoire, Position)

Interface de la couche articulaire

- Compétences de la position
  Actuelle_pos(Position)
  Valide_pos(Position)
- Compétence de la home position
  Reference(Position)
  Actuelle_Home(Position)
  Nest

...

- Compétence de la vitesse
  valide_vit(Vitesse)
  actuelle_ vit(Vitesse)
- Compétence du mouvement
  dernier(Mouvement)
  immédiat(Mouvement)
  vitesse-mvt(Mouvement, Vitesse)
  destination_ mvt(Mouvement, Position)
  valide_mvt(Mouvement)

- Compétence de la trajectoire
  parcourir_traj (Traj)
  destination_traj(Traj, Position)
  valide_traj(Traj)
- Compétence de la force
  valide_force(Force)
  fermer_force(Force)
  ouvrir_force(Force)
- Compétence du temps
  valide_tps(Temps)
  attendre(Temps)

STRUCTURE DE DONNEE DESOBJETS DEFINIS DANS L'INTERFACE DE LA COUCHE DE NIVEAU ACTIONNEUR

---

ROBOT ARTICULAIRE : n'est qu'un concept.
Il n'est pas implémenté sous la forme d'une structure de donnée.

LA POSITION : Coord (A1, A2, A3, A4, A5)
Les Ai représentent respectivement les angles de rotation du corps et de l'épaule, l'inclinaison du poignet et la torsion du poignet. Ces angles sont exprimés par des entiers en 100ème de degrés.

LA FORCE : Force (P1, P2, T)
P1, P2, et T sont des entiers. P1 et P2 représentent une pression de la pince exprimée dans une échelle de 0 à 7. T est une durée exprimée en 10è de secondes et représente la durée pendant laquelle la pression P1 est exercée. Après cette durée, la pression P2 est exercée.
Cette durée s'exprime dans une échelle de 0 à 99.

LA VITESSE : Vit (v)
v exprime la vitesse de translation de la pince dans l'espace de travail du robot et est exprimée en mm par seconde. Cette vitesse est exprimée à l'aide d'un entier qui varie entre 40 et 400.

LE MOUVEMENT : Mvt (Vit, Pos)
Vit est une structure de donnée du type vitesse. Pos est une structure de donnée du type position.

LA TRAJECTOIRE : Traj (|mvt1, mvt 2....mvt n|)
La trajectoire est une liste de structures du type mouvement.

LE TEMPS : t(T)
T est une entier positif qui représente un temps exprimé en 1/10 de secondes.

```
/*
IMPLEMENTATION DE LA COUCHE ARTICULAIRE
_______________________________________

Competences du mouvement
________________________

/*
DERNIER(Mouvement)
------------------
*/

dernier(mvt(Vit,Coord)) :-
          current_pos(Coord),
          speed(Vit),!.

/*
IMMEDIAT (Mouvement)
--------------------
 */
immediat(mvt(Vit,Coord)) :-
          valide(mvt(Vit,Coord)),
          set_speed(Vit),
          move_immediate(Coord),!.


VALIDE_MVT (Mouvement)
----------------------
*/

valide_mvt(mvt(vit(V),P)) :-
     valide_pos(P),
     valide_vit(vit(V)),
     V =< 400,
     V >= 40,!.
/*

VITESSE_MVT(Mouvement,Vitesse).
-------------------------------
*/

vitesse_mvt(mvt(V,_),V).

/*
DESTINATION_MVT(Mouvement,Position).
------------------------------------
*/

destination_mvt(mvt(_,P),P).

/*
Competence de la trajectoire
____________________________

PARCOURIR_TRAJ(Trajectoire)
---------------------------
```

```
*/

parcourir_traj(traj(T)) :-
          non_vide(traj(T)),!,
          decoupe(T,Traj_douce),
          parcourir_traj_douce(Traj_douce),!.

parcourir_traj_douce([]).
parcourir_traj_douce([Segment|R]) :-
          parcourir_segment(Segment),
          parcourir_traj_douce(R).

parcourir_segment(Segment):-
          repartition(Segment,Trajectoire_continue),
          parcourir_traj_continue(Trajectoire_continue).

parcourir_traj_continue([]).
parcourir_traj_continue([Segment_continu|R]):-
          parcours_continu(Segment_continu),
          parcourir_traj_continue(R).

parcours_continu([]):- go.
parcours_continu([mvt(Vit,Pos)|R]):-
          valide(mvt(Vit,Pos)),
          set_speed(Vit),
          set_position(Pos),
          parcours_continu(R).

non_vide(traj([_|_])).


/*
DESTINATION_TRAJ(Trajectoire,Position).
---------------------------------------
*/

destination_traj(traj([mvt(_,Pos)]),Pos).
destination_traj(traj([M|R]),Pos):- destination(traj(R),Pos).


/*
Competence de la position
_________________________

NEST
----
*/

nest :- nt.

/*
ACTUELLE_POS(Position)
----------------------
*/

actuelle_pos(X) :- current_pos(X),!.
```

```
/*

VALIDE_POS(Position)
----------------
*/

valide_pos(coord(A1,A2,A3,A4,A5)) :-
     integer(A1),
     integer(A2),
     integer(A3),
     integer(A4),
     integer(A5),
     current_home(coord(H1,H2,H3,H4,H5)),
     A1 >=  ~H1,          A1 =< 30000 - H1,
     A2 >=  ~13000 - H2, A2 =< ~H2,
     A3 >=  ~H3,          A3 =< 9000  - H3,
     A4 >=  ~H4,          A4 =< 18000 - H4,
     A5 >=  ~18000 - H5, A5 =< 18000 - H5,!.

/*
REFERENCE(Position).
-------------------
*/

reference(coord(A1,A2,A3,A4,A5)) :-
          valide(coord(A1,A2,A3,A4,A5)),
          set_home(coord(A1,A2,A3,A4,A5)),!.

/*
ACTUELLE_HOME(Position)
----------------------
*/

actuelle_home(Position) :- current_home(Position).

/*
Competence de la force
______________________

FERMER (Force)
--------------
*/

fermer(force(X,Y,T)) :-
     valide-force(force(X,Y,T)),
     set_grip_force(force(X,Y,T)),
     grasp,!.

/*
OUVRIR(Force)
-------------
*/

ouvrir(force(X,Y,T)) :-
     valide-force(force(X,Y,T)),
     set_grip_force(force(X,Y,T)),
     release,!.
```

```
/*
VALIDE_FORCE(Force)
-------------------
*/

valide_force(force(F1,F2,T)):-
     integer(F1),
     integer(F2),
     integer(T),
     F1 =< 9,   F1 >= 0,
     F2 =< 9,   F2 >= 0,
     T =< 99,   T >=  0,!.

/*
Competence du temps
___________________

ATTENDRE (Temps)
----------------
*/

attendre(t(T)) :-
          valide_tps(t(T)),
          set_time(T),!.

VALIDE_TPS (Temps)
--------------
*/

valide_tps(t(Temps)) :- integer(Temps).
```

```
/*
MODULE DE GENERATION DE MESSAGES DE MESSAGES
```

Ce module est spécifiquement chargé d'envoyer des commandes à l'unité de commande du robot. La syntaxe de ces commandes est définie en Annexe 4.

```
NT
--
*/

nt :-     current_home(coord(A1,A2,A3,A4,A5)), /* le robot a deja ete initialise */
          A6 is ~A1, A7 is ~A2,
          A8 is ~A3, A9 is ~A4,
          A10 is ~A5,
          retract(ctr_pos(_)),
          assert(ctr_pos(0)),
          set_position(coord(A6,A7,A8,A9,A10)),
          go,
          telling(P),
          current_output(Out),
          tell(Out),
          printlist(['NT']),
          tell(P),!.


nt   :-   telling(P),
          current_output(Out),
          tell(Out),
          printlist(['NT']),                    /* le robot est initialise  */
          assert(speed(vit(40))),               /* pour la premiere fois    */
          assert(ctr_pos(0)),
          assert(current_home(coord(0,0,0,0,0))), /* default home position   */
          assert(grip_force(force(5,0,10))),
          assert(current_pos(coord(0,0,0,0,0))),
          tell(P).

/*
SET POSITION.        enregistrement d'une position.
------------
*/

set_position(coord(T1,T2,T3,T4,T5)) :-
          position(X),
          rec_pos(X,coord(T1,T2,T3,T4,T5)),!.


rec_pos(1,coord(T1,T2,T3,T4,T5)):-
          retract(current_pos(coord(T6,T7,T8,T9,T10))),
          assert(current_pos(coord(T1,T2,T3,T4,T5))),
          conversion_en_pas(coord(T1,T2,T3,T4,T5),pas(P1,P2,P3,P4,P5)),
          conversion_en_pas(coord(T6,T7,T8,T9,T10),pas(P6,P7,P8,P9,P10)),
          position(_),
          telling(P),
          current_output(Out),
          tell(Out),
          printlist(['PS',1,P6,P7,P8,P9,P10,0]),
          printlist(['PS',2,P1,P2,P3,P4,P5,0]),
          tell(P),!.
```

```
rec_pos(X,coord(T1,T2,T3,T4,T5)):-
        conversion_en_pas(coord(T1,T2,T3,T4,T5),pas(P1,P2,P3,P4,P5)),
        retract(current_pos(_)),
        assert(current_pos(coord(T1,T2,T3,T4,T5))),
        telling(P),
        current_output(Out),
        tell(Out),
        printlist['PS',X,P1,P2,P3,P4,P5,0]),
        tell(P),!.

position(X) :-
        retract(ctr_pos(Y)),
        X is Y + 1,
        assert(ctr_pos(X)),!.

/*
MOVE IMMEDIATE
--------------
*/
```

enregistrement d'une position et déplacement des organes du SMA à cette position.

```
move_immediate(coord(T1,T2,T3,T4,T5)) :-
        conversion_en_pas(coord(T1,T2,T3,T4,T5),pas(P1,P2,P3,P4,P5)),
        retract(current_pos(_)),
        assert(current_pos(coord(T1,T2,T3,T4,T5))),
        telling(P),
        current_output(Out),
        tell(Out),
        printlist['PS',1,P1,P2,P3,P4,P5,0]),
        printlist['MO 1']),
        tell(P),
        retract(ctr_pos(_)),
        assert(ctr_pos(0)),!.

/*
SET HOME
--------
*/
```

Enregistrement de la "home position"

```
set_home(coord(T1,T2,T3,T4,T5)) :-
        retract(current_pos(coord(A1,A2,A3,A4,A5))),
        C1 is A1 - T1,                          /* reajustement de la position  */
        C2 is A2 - T2,                          /* de la position courrante     */
        C3 is A3 - T3,                          /* du bras articule en fonction */
        C4 is A4 - T4,                          /* de la nouvelle home position */
        C5 is A5 - T5,
        assert(current_pos(coord(C1,C2,C3,C4,C5))),
        conversion_en_pas(coord(T1,T2,T3,T4,T5),pas(P1,P2,P3,P4,P5)),
        telling(P),
        current_output(Out),
        tell(Out),
        printlist['PS',0,P1,P2,P3,P4,P5,0]),
        tell(P),
        retract(current_home(coord(D1,D2,D3,D4,D5))),
        B1 is D1 + T1,                          /* Calcul de la position de la  */
        B2 is D2 + T2,                          /* 'Home position' par rapport  */
        B3 is D3 + T3,                          /* a l'origine mecanique du SMA */
        B4 is D4 + T4,
        B5 is D5 + T5,
        assert(current_home(coord(B1,B2,B3,B4,B5))),!.
```

```
/*
GO
--
*/
```

passage successif par toutes les positions enregistrées dans la mémoire de l'unité de commande.

```
go:-    retract(ctr_pos(1)),          /* invoque s'il n'y a qu'une seule   */
        telling(P),                   /* position a parcourir              */
        current_output(Out),
        tell(Out),
        printlist(['MO 1']),
        printlist(['PC 1']),
        tell(P),
        assert(ctr_pos(0)),!.

go:-    retract(ctr_pos(Y)),          /* invoque s'il y a plus */
        assert(ctr_pos(0)),           /* d'une position a parcourir */
        telling(P),
        current_output(Out),
        tell(Out),
        printlist(['MO 1']),
        S is Y - 1,
        printlist(['MC',S]),
        printlist(['PC 1']),
        tell(P),!.
```

```
/*
SET SPEED
---------
*/
```

Modification de la vitesse de déplacement des corps d'une position à l'autre

```
set_speed(vit(S)) :-
        speed(vit(D)),
        S / 40 =:= D / 40,!.

set_speed(vit(S)) :-
        not speed(S),
        telling(P),
        current_output(Out),
        tell(Out),
        C is (S + 20) / 40 - 1,
        printlist(['SP',C]),
        tell(P),
        retract(speed(_)),
        assert(speed(S)),!.


set_speed(vit(S)).
```

```
/*
GRASP
-----
*/
```

fermeture de la pince

```
grasp :-  telling(P),
        current_output(Out),
        tell(Out),
        printlist(['GC']),
        printlist(['GF 1']),
        tell(P),!.
```

```
/*
RELEASE                 ouverture de la pince
-------
*/

release:- telling(P),
          current_output(Out),
          tell(Out),
          printlist(['GO']),
          printlist(['GF 0']),
          tell(P),!.

/*
SET GRIP FORCE          modification de la force de serrage ou d'ouverture
--------------          de la pince
*/

set_grip_force(X) :-
          grip_force(X),!.

set_grip_force(force(F1,F2,Temps)) :-
          not grip_force(force(F1,F2,Temps)),
          telling(P),
          current_output(Out),
          tell(Out),
          printlist(['GP',F1,F2,Temps]),
          tell(P),
          retract(grip_force(_)),
          assert(grip_force(force(F1,F2,Temps))),!.

/*
SET TIME.               blocage de l'activité de l'interpreteur de
---------               commandes de l'unité de commande pendant
*/                      un temps donné.

set_time(Temps) :-
          Temps > 99,
          T is Temps - 99,
          telling(P),
          current_output(Out),
          tell(Out),
          printlist(['TI',99]),
          tell(P),
          set_time(T),!.

set_time(Temps) :-
          Temps =< 99,
          telling(P),
          current_output(Out),
          tell(Out),
          printlist(['TI',Temps]),
          tell(P),!.
/*
SET OUTPUT              Selection de la sortie
----------
*/

set_output(X) :-
          retract(current_output(_)),
          assert(current_output(X)),
          retractall(speed(_)),
          retractall(ctr_pos(_)),
          retractall(tcp_actuel(_)),
          retractall(current_home(_)),
          retractall(grip_force(_)),
          retractall(current_pos(_)),!.
```

```
printlist([X]):- integer(X),printnum(X),nl,!.
printlist([X]):- write(X),nl,!.
printlist([X1|X2]):-
          integer(X1),
          printnum(X1),
          put(44),
          printlist(X2),!.
printlist([X1|X2]):-
          write(X1),
          write(' '),
          printlist(X2).

printnum(X):- X<0,!,
          put(45),
          Y is ~X,
          write(Y).
printnum(X):- write(X).

conversion_en_pas(coord(T1,T2,T3,T4,T5),pas(P1,P2,P3,P4,P5)):-
          P1 is ~T1 * 4 / 10,
          P2 is T2 * 4 / 10,
          P3 is T3 * 4 / 10,
          A  is T4 * 4 / 30,
          B  is T5 * 4 / 30,
          P4 is A + B,
          P5 is B - A.

current_output(printer).
```

DEUXIEME COMPETENCE DE LA TRAJECTOIRE : La découpe

Cette compétence est développée dans un module à part (module découpe) pour des raisons de modifiabilité

```
/*
DECOUPE (TRAJECTOIRE, TRAJECTOIRE DOUCE )
-----------------------------------------

*/

decoupe([X],[[X]]) :-!.
decoupe(T,Y) :-
          current_pos(C),
          speed(S),
          dec(mvt(S,C),T,T,Y).

dec(_,[X],L,[L]):- !.

dec(C,[T1,T2|T],L,[Y|Z]):-
          inversion(C,T1,T2),
          trouver_liste(L,[T2|T],Y),
          dec(T1,[T2|T],[T2|T],Z),!.

dec(_,[T1,T2|T],L,Y) :- dec(T1,[T2|T],L,Y).

inversion(Pos1,Pos2,Pos3) :-
          sens_rotation(Pos1,Pos2,Rot1),
          sens_rotation(Pos2,Pos3,Rot2),
          choc(Rot1,Rot2).

trouver_liste( Y,Y,[]):-!.
trouver_liste([X|Z],Y,[X|T]) :- trouver_liste(Z,Y,T).

sens_rotation(mvt(_,coord(P11,P12,P13,P14,P15)),
              mvt(_,coord(P21,P22,P23,P24,P25)),
              [R1,R2,R3,R4,R5]) :-
           R1 is P11 - P21,
           R2 is P12 - P22,
           R3 is P13 - P23,
           R4 is P14 - P24,
           R5 is P15 - P25.

choc([X1|X],[Y1|Y]) :- sens_oppose(X1,Y1),!.
choc([X1|X],[Y1|Y]) :- choc(X,Y).

sens_oppose(X1,X2) :-
          X1 < 0,
          X2 > 0.
sens_oppose(X1,X2) :-
          X1 > 0,
          X2 < 0.
```

COMPETENCE DU SEGMENT : La répartition

Implémentée à part dans le module de répartition.

```
/*
REPARTITION(Segment,Trajectoire_continue)
_____________________________________
*/
repartition(X,Z) :-
     caracteristiques_trajectoire(X,Long,Vit,Nombre),
     maximum_positions(MP),
     suite_elements_continus(X,Long,Vit,Nombre,MP,Y),
     recoller_morceaux(Y,Z,MP),!.

suite_elements_continus(X,Long,_,_,MP,[X]) :-
          Long =< MP,!.

suite_elements_continus(X,Long,_,Long,MP,Y) :-
          suite_elements_cont_taille_max(X,Y,MP),!.

suite_elements_continus(X,_,Vit,_,MP,Y) :-
          couper_a_vitesse_faible(X,Vit,T1,T2),
          caracteristiques_trajectoire(T1,Long1,Vit1,Nombre1),
          suite_elements_continus(T1,Long1,Vit1,Nombre1,MP,U1),
          caracteristiques_trajectoire(T2,Long2,Vit2,Nombre2),
          suite_elements_continus(T2,Long2,Vit2,Nombre2,MP,U2),
          append2(U1, U2, Y).

caracteristiques_trajectoire(X,Long,Vit,Nombre) :-
          info_trajectoire(0,10,0,X,Long,Vit,Nombre).

info_trajectoire(Long, Vit, N, [], Long, Vit, N).
info_trajectoire(L1, V1, N1, [pos(vit(V),_)|T], X, Y, Z) :-
          V1 == V,!,
          L2 is L1 + 1,
          N2 is N1 + 1,
          info_trajectoire(L2, V1, N2, T, X, Y, Z).
info_trajectoire(L1, V1, N1, [pos(vit(V),_)|T], X, Y, Z) :-
          V < V1, !,
          L2 is L1 + 1,
          info_trajectoire(L2, V, 1, T, X, Y, Z).
info_trajectoire(L1, V, N, [_|T], X, Y, Z) :-
          L2 is L1 + 1,
          info_trajectoire(L2, V, N, T, X, Y, Z).

recoller_morceaux([],[],_).
recoller_morceaux([X],[X],_).
recoller_morceaux([X,Y|T],U, MP) :-
          longueur(X,L1),
          longueur(Y,L2),
          MP >= L1 + L2,
          append2(X,Y,V),
          recoller_morceaux([V|T],U, MP).
recoller_morceaux([X,Y|T],[X|U], MP) :-
          recoller_morceaux([Y|T],U, MP).

suite_elements_cont_taille_max(X,[Element_continu|R],MP) :-
          append2(Element_continu, Reste,X),
          longueur(Element_continu,MP),
          suite_elements_cont_taille_max(Reste, R, MP),!.
```

```
suite_elements_cont_taille_max([],[],_):-!.      /* Le Reste est une liste vide */
suite_elements_cont_taille_max(X,[X], _).        /* Taille du Reste < MP        */

couper_a_vitesse_faible(Traj, Vit, T1, [pos(vit(Vit),Coord)|T2]) :-
          append2(T1, [pos(vit(Vit),Coord)|T2], Traj),
          evaluer(T1, [pos(vit(Vit),Coord)|T2], Long_premier, Evaluation_actuel),
          Evaluation_actuel > 0,
          evaluer_suivant(Long_premier, T2, Vit, Evaluation_suivant),
          Evaluation_suivant < Evaluation_actuel,
          !.
couper_a_vitesse_faible([T1|Traj], _, [T1], Traj ).

evaluer(T1, T2, L1, E) :-
          longueur(T1, L1),
          longueur(T2, L2),
          minimum(L1, L2, E).

evaluer_suivant(Longueur, Traj, Vit, Eval) :-
          append2(X, [pos(vit(Vit),_)|Y], Traj),
          longueur(X, L1),
          longueur([pos(vit(Vit),_)|Y], L2),
          L3 is Longueur + 1 + L1,
          minimum(L3, L2, Eval), !.
evaluer_suivant(_, _, _, 0).                     /* Pas de solution suivante */

minimum(L1, L2, L1):-
          L1 =< L2, !.
minimum(L1, L2, L2):-
          L1 > L2.
append2([], X, X).
append2([X|Y], U, [X|Z]) :-
          append2(Y, U, Z).

longueur([],0) :- !.
longueur([X|Y],L) :-
          longueur(Y,T),
          L is T + 1.

maximum_positions(629).
info_traj(0,0,0).
```

# ANNEXE C

## EXTENSION DU LANGAGE

## IMPLEMENTATION DE LA CINEMATIQUE DIRECTE

## IMPLEMENTATION DE LA CINEMATIQUE INVERSE

/*
EXTENSION DU LANGAGE PROLOG

Dans le prolog de W.F. Clocksin & C.S. Mellish, toute operation arithmetique se fait toujours en nombre entier. Pour des applications en robotique ou en calcul numerique, les mombres entiers s'averent tres peu adaptes. Pour ce genre de travaux , nous avons ajoute une extension au langage qui permet de travailler avec des nombres reels.
Le nombre reel est represente par une structure de donnees identifiee par le mot cle 'real'. La structure 'real' comporte deux nombres entiers , repectivement la mantisse et l'exposant.

real(Mantisse,Exposant)

Un tel type pose certains problemes :

- le probleme de la vitesse : Nous savons deja que les operations arithmetiques en virgule flottante coutent cher en temps de calcul. En prolog, les operations arithmetiques en nombre reel coutent encore plus cher, en effet, il faut ajouter au temps de calcul un temps de conversion et un temps de normalisation.
  La coversion est une operation effectuee par l'interpreteur qui consiste a transformer la representation real(Mantisse,Exposant), en un nombre en virgule flottante.
  La normalisation est l'operation inverse. Elle est effectuee par l'interpreteur en fin de calcul.

- Le probleme du "matching" : Le nombre reel ne peut representer qu'un nombre limite de chiffres. Les erreurs d'arrondis qui en resultent posent dans certains cas un probleme de matching.

  exemple: arcsin(sin(X)) <> X a cause des erreurs d'arromdis, Le matching ne sera des lors plus possible entre X et arcsin(sin(X)).

---

Les nouveaux predicats builtin

| | |
|---|---|
| comvrt(Mant,Exp,Int) | Conversion des nombres reels en nombres entiers. Ce predicat calcule le nombre entier correspondant a l'arrondis du nombre reel represente par la notation Mant,Exp. |
| norm(Int,Mant,Exp) | Conversion des nombres entiers en nombres reels. Le predicat determine la representation Mantisse,Exposant correspondant au nombre entier donne. |
| addrl(M1,E1,M2,E2,M3,E3) | Calcule l'addition de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. le nombre represente par M3,E3 est le resultat de l'operation d'addition. |
| subrl(M1,E1,M2,E2,M3,E3) | Effectue la soustraction de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. le nombre represente par M3,E3 est le resultat de l'operation de soustraction. |

| | |
|---|---|
| divr1(M1,E1,M2,E2,M3,E3) | Effectue la division de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. le nombre represente par M3,E3 est le resultat de l'operation de division,M2,E2 represente un diviseur non nul. |
| multr1(M1,E1,M2,E2,M3,E3) | Effectue la multiplication de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. le nombre represente par M3,E3 est le resultat de l'operation de multiplication. |
| sqrout(M1,E1,1,M2,E2) | Calcule le carre du nombre reel represente par M1,E1. le resultat de l'elevation au carre est represente par M2,E2. |
| sqrout(M1,E1,2,M2,E2) | Calcule la racine carree du nombre reel represente par M1,E1. le resultat de l'extraction de la racine carree est represente par M2,E2. |
| trigono(M1,E1,1,M2,E2) | calcule le sinus de l'angle represente par M1,E1. |
| trigono(M1,E1,4,M2,E2) | calcule l'arc dont le sinus est represente par M1,E1. |
| trigono(M1,E1,2,M2,E2) | calcule le cosinus de l'angle represente par M1,E1. |
| trigono(M1,E1,5,M2,E2) | calcule l'arc dont le cosinus est represente par M1,E1. |
| trigono(M1,E1,3,M2,E2) | calcule la tangente de l'angle represente par M1,E1. |
| trigoro(M1,E1,5,M2,E2) | calcule l'arc dont la tangente est represente par M1,E1. |

Predicats definis a l'aide des nouveaux predicats builtin

---

| | |
|---|---|
| sumr(R1,R2,R3) | Permet de calculer la somme ou la difference de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. Le predicat affecte une valeur au couple d'arguments Mantisse,Exposant non instancies de maniere a etablir l'egalite N1 + N2 = N3. |
| timesr(R1,R2,R3) | Permet de calculer le produit ou le quotient de deux nombres reels representes a l'aide de la notation Mantisse,Exposant. Le predicat affecte une valeur au couple d'arguments Mantisse,Exposant non instancies de maniere a etablir l'egalite N1 / N2 = N3. |
| sin(N1,N2) | Permet de calculer un sinus ou un arc sinus. ce predicat etablit la relation N2 = sin(N1) ou N1 = arcsin(N2). |
| cos(N1,N2) | Permet de calculer un cosinus ou un arc cosinus. ce predicat etablit la relation N2 = cos(N1) ou N1 = arcos(N2). |
| tan(N1,N2) | Permet de calculer la tagente d'un angle ou de calculer l'arc dont on connait la tangente. ce predicat etablit la relation N2 = sin(N1) ou N1 = arcsin(N2). |

```
Conversion nombre entier <---> nombre reel
------------------------------------------
*/

int(real(Mant,Exp),Int):-
     integer(Mant),
     integer(Exp),!,
     convrt(Mant,Exp,Int).
int(real(Mant,Exp),Int):-
     integer(Int),
     norm(Int,Mant,Exp).
/*
Operations arthmetiques sur les reels
-------------------------------------
*/

sumr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant1),
     integer(Exp1),
     integer(Mant2),
     integer(Exp2),!,
     addrl(Mant1, Exp1, Mant2, Exp2, Mant3, Exp3).
sumr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant1),
     integer(Exp1),
     integer(Mant3),
     integer(Exp3),!,
     subrl(Mant3, Exp3, Mant1, Exp1, Mant2, Exp2).
sumr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant2),
     integer(Exp2),
     integer(Mant3),
     integer(Exp3),!,
     subrl(Mant3, Exp3, Mant2, Exp2, Mant1, Exp1).

timesr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant1),
     integer(Exp1),
     integer(Mant2),
     integer(Exp2),!,
     multrl(Mant1, Exp1, Mant2, Exp2, Mant3, Exp3).
timesr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant1),
     integer(Exp1),
     integer(Mant3),
     integer(Exp3),!,
     Mant1 \== 0,
     divrl(Mant3, Exp3, Mant1, Exp1, Mant2, Exp2).
timesr(real(Mant1, Exp1), real(Mant2, Exp2), real(Mant3, Exp3)) :-
     integer(Mant2),
     integer(Exp2),
     integer(Mant3),
     integer(Exp3),!,
     Mant2 \== 0,
     divrl(Mant3, Exp3, Mant2, Exp2, Mant1, Exp1).

sqr(real(M1,E1),real(M2,E2)) :-
     integer(M2),
     integer(E2),
     M2 >= 0,!,
     sqrout(M2,E2,1,M1,E1).
sqr(real(M1,E1),real(M2,E2)) :-
     integer(M1),
     integer(E1),!,
     sqrout(M1,E1,2,M2,E2).
```

```
valeur_absolue(real(M1,E),real(M2,E)) :-
     integer(M1),
     M1 < 0,
     M2 is ~M1,!.

valeur_absolue(real(M1,E),real(M2,E)) :-
     integer(M2),
     M2 < 0,
     M1 is ~M1,!.

valeur_absolue(real(M,E),real(M,E)).

signe_oppose(real(M1,E),real(M2,E)) :-
     nonvar(M1),
     M2 is ~M1,!.

signe_oppose(real(M1,E),real(M2,E)) :-
     nonvar(M2),
     M1 is ~M2.


/*
Fonctions trigonometriques
--------------------------
*/

sin(X,real(_,E)) :- var(X), E > 1,!,fail.
sin(X,real(M,1)) :- var(X), M > 100000000,!,fail.
sin(X,real(M,1)) :- var(X), M < ~100000000,!,fail.
sin(real(M1,E1),real(M2,E2)) :-
     integer(M1),
     integer(E1),!,
     trigono(M1,E1,1,M2,E2).
sin(real(M1,E1),real(M2,E2)) :-
     integer(M2),
     integer(E2),!,
     trigono(M2,E2,4,M1,E1).
cos(X,real(_,E)) :- var(X), E > 1,!,fail.
cos(X,real(M,1)) :- var(X), M > 100000000,!,fail.
cos(X,real(M,1)) :- var(X), M < ~100000000,!,fail.
cos(real(M1,E1),real(M2,E2)) :-
     integer(M1),
     integer(E1),!,
     trigono(M1,E1,2,M2,E2).
cos(real(M1,E1),real(M2,E2)) :-
     integer(M2),
     integer(E2),!,
     trigono(M2,E2,5,M1,E1).

tan(real(M1,E1),real(M2,E2)) :-
     integer(M1),
     integer(E1),!,
     real(M1,E1) \== real(900000000, 2),
     real(M1,E1) \== real(270000000, 3),
     trigono(M1,E1,3,M2,E2).
tan(real(M1,E1),real(M2,E2)) :-
     integer(M2),
     integer(E2),!,
     trigono(M2,E2,6,M1,E1).
```

```
procedure Normalisation(Num : real ; var Mant, Exp:longint);
(* normalisation de la mantisse et ajustement de l'exposant *)
begin
   Exp := 0;
   while Abs(Num) >= 1 do begin
        Num := Num / 10;
        Exp  := Exp + 1
   end;
   while (Abs(Num) < 0.1) and (Num <> 0) do begin
        Num := Num * 10;
        Exp  := Exp - 1
   end;
   Mant := trunc(Num * 1000000000);
   if Mant = 0 then Exp := 0
end (* Normalisation *);

procedure Conversion(Mant,Exp: longint;var Num : real);

begin
    Num := Mant / Pwroften(9 - Exp)
end;(* Conversion *)
                                    (* predicat NORM *)
procedure Donorm;
(* transforme un nombre entier en un nombre en virgule
    flottante normalisee *)
var R : real;
    Mant, Exp : longint;
begin
    R := evaluate(argval[1], 0);
    Normalisation(R, Mant, Exp);
    result := IntResult(argval[2], callenv, Mant);
    result := IntResult(argval[3], callenv, Exp)
end;
                                    (* predicat convrt *)
procedure Doconvrt;
(* transforme un nombre en virgule flottante normalise en
   nombre entier *)
var Mant, Exp : longint;
    R : real;
begin
    Mant := evaluate(argval[1],0);
    Exp := evaluate(argval[2],0);
    if Exp <= 0 then result := IntResult(argval[3],callenv,0)
    else begin
       Conversion (Mant,Exp,R);
       result := IntResult(argval[3],callenv,trunc(R))
    end
end;
                                    (* predicat addrl *)
procedure DoAddreal;
(* addition de deux nombres en virgule flottante *)
var Mant,Exp : longint;
    R1,R2 : real;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    conversion(evaluate(argval[3],0),evaluate(argval[4],0),R2);
    Normalisation(R1 + R2 ,Mant, Exp);
    result := IntResult(argval[5],callenv,Mant);
    result := IntResult(argval[6],callenv,Exp)
end;
```

```
procedure DoSubreal;        (* predicat  sobrl *)
(* soustraction de deux nombres en virgule flottante *)
var Mant,Exp : longint;
    R1,R2 : real;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    conversion(evaluate(argval[3],0),evaluate(argval[4],0),R2);
    Normalisation(R1 - R2 ,Mant, Exp);
    result := IntResult(argval[5],callenv,Mant);
    result := IntResult(argval[6],callenv,Exp)
end;

procedure DoMultreal;    (* predicat  multrl *)
(* multiplication de deux nombres en virgule flottante *)
var Mant,Exp : longint;
    R1,R2 : real;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    conversion(evaluate(argval[3],0),evaluate(argval[4],0),R2);
    Normalisation(R1 * R2 ,Mant, Exp);
    result := IntResult(argval[5],callenv,Mant);
    result := IntResult(argval[6],callenv,Exp)
end;

procedure DoDivreal;    (* predicat  divrl *)
(* division de deux nombres en virgule flottante *)
var Mant,Exp : longint;
    R1,R2 : real;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    conversion(evaluate(argval[3],0),evaluate(argval[4],0),R2);
    Normalisation(R1 /  R2 ,Mant, Exp);
    result := IntResult(argval[5],callenv,Mant);
    result := IntResult(argval[6],callenv,Exp)
end;

procedure DoSqrout;        (* predicat  sqrout *)
(* calcul du carre ou de la racine carree d'un nombre *)
var Mant,Exp : longint;
    R1, R2 : real;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    case evaluate(argval[3],0) of
    1: R2 := Sqrt(R1);
    2: R2 := Sqr(R1);
    end;
    Normalisation(R2 ,Mant, Exp);
    result := IntResult(argval[4],callenv,Mant);
    result := IntResult(argval[5],callenv,Exp)
end;
```

```
procedure Dotrigono;          (* Predical trigono *)
const PI = 3.14159265;
var R1, R2, R3, R4 : Real;
    Mant,Exp :longint;
begin
    conversion(evaluate(argval[1],0),evaluate(argval[2],0),R1);
    case evaluate(argval[3],0) of
    1: R2 := sin(R1 / 180 * PI);
    2: R2 := cos(R1 / 180 * PI);
    3: begin (fonction tangeante)
          R4 := R1 / 180 * PI;
          R2 := sin(R4)/cos(R4)
       end;
    4: begin (fonction arcsin)
          if R1 = 0 then R2 := 0
          else R2 := arctan(R1 / sqrt( 1 - sqr(R1))) * 180 /PI
       end;
    5: begin (fonction arcos)
          if R1 = 0 then R2 := 90
          else R2 := arctan(sqrt( 1 - sqr(R1)) / R1) * 180 /PI
       end;
    6: R2 := Arctan(R1) *180 / PI;
    end;
    normalisation(R2,Mant,Exp);
    result := IntResult(argval[4],callenv,Mant);
    result := IntResult(argval[5],callenv,Exp)
end;
```

```
/*
MODULE DE LA CINEMATIQUE

_____________________________
*/

transformation(coord(A1,A2,A3,A4,A5),tcp(X1,Y1,Z1,Tg,A5,A1)) :-
     conversion_en_reel2(A1,A2,A3,A4,A5,T1,T2,T3,T4,T5),
     signe_oppose(T2,TT2),
     signe_oppose(T3,TT3),
     signe_oppose(T4,TT4),
     calcul_valeur_X(T1,TT2,TT3,TT4,X),
     calcul_valeur_Y(T1,TT2,TT3,TT4,Y),
     calcul_valeur_Z(T1,TT2,TT3,TT4,Z),
     calcul_valeur_tangage(T1,T2,T3,T4,T),
     conversion_en_entier2(X,Y,Z,T,X1,Y1,Z1,Tg).!.


calcul_valeur_X(T1,T2,T3,T4,X):-

/*      X :=  L2C1C2 + L3C1C2C3 - L3C1S2S3 + L4C1C2C3C4
           - L4C1S2S3C4 - L4C1C2S3S4 - L4C1S2C3S4.      */

     cos(T1,C1),          cos(T2,C2),
     sin(T2,S2),          cos(T3,C3),
     sin(T3,S3),          cos(T4,C4),
     sin(T4,S4),
     timesr(C1,C2,C1C2), timesr(C3,C4,C3C4),
     timesr(C1,S2,C1S2), timesr(C3,S4,C3S4),
     timesr(S3,C4,S3C4), timesr(S3,S4,S3S4),
     timesr(real(220000000,3),C1C2,L2C1C2),
     timesr(C1C2,C3,C1C2C3),
     timesr(real(160000000,3),C1C2C3,L3C1C2C3),
     timesr(C1S2,S3,C1S2S3),
     timesr(real(160000000,3),C1S2S3,L3C1S2S3),
     timesr(C1C2,C3C4,C1C2C3C4),
     timesr(real(204600000,3),C1C2C3C4,L4C1C2C3C4),
     timesr(C1S2,S3C4,C1S2S3C4),
     timesr(real(204600000,3),C1S2S3C4,L4C1S2S3C4),
     timesr(C1C2,S3S4,C1C2S3S4),
     timesr(real(204600000,3),C1C2S3S4,L4C1C2S3S4),
     timesr(C1S2,C3S4,C1S2C3S4),
     timesr(real(204600000,3),C1S2C3S4,L4C1S2C3S4),
     sumr(L2C1C2,L3C1C2C3,U1),
     sumr(L3C1S2S3,U2,U1),
     sumr(L4C1C2C3C4,U2,U3),
     sumr(L4C1S2S3C4,U4,U3),
     sumr(L4C1C2S3S4,U5,U4),
     sumr(L4C1S2C3S4,X,U5).

calcul_valeur_Y(T1,T2,T3,T4,Y) :-

/*       Y :=   L2S1C2 + L3S1C2C3 - L3S1S2S3 + L4S1C2C3C4
            - L4S1S2S3C4 - L4S1C2S3S4 - L4S1S2C3S4.      */

     cos(T2,C2),          sin(T2,S2),
```

```
      cos(T3,C3),          sin(T3,S3),
      cos(T4,C4),          sin(T4,S4),
      sin(T1,S1),
      timesr(C3,C4,C3C4), timesr(C3,S4,C3S4),
      timesr(S1,C2,S1C2), timesr(S3,C4,S3C4),
      timesr(S1,S2,S1S2), timesr(S3,S4,S3S4),
      timesr(real(220000000,3),S1C2,L2S1C2),
      timesr(S1C2,C3,S1C2C3),
      timesr(real(160000000,3),S1C2C3,L3S1C2C3),
      timesr(S1S2,S3,S1S2S3),
      timesr(real(160000000,3),S1S2S3,L3S1S2S3),
      timesr(S1C2,C3C4,S1C2C3C4),
      timesr(real(204600000,3),S1C2C3C4,L4S1C2C3C4),
      timesr(S1S2,S3C4,S1S2S3C4),
      timesr(real(204600000,3),S1S2S3C4,L4S1S2S3C4),
      timesr(S1C2,S3S4,S1C2S3S4),
      timesr(real(160000000,3),S1C2S3S4,L4S1C2S3S4),
      timesr(S1S2,C3S4,S1S2C3S4),
      timesr(real(160000000,3),S1S2C3S4,L4S1S2C3S4),
      sumr(L2S1C2,L3S1C2C3,U6),
      sumr(L3S1S2S3,U7,U6),
      sumr(L4S1C2C3C4,U7,U8),
      sumr(L4S1S2S3C4,U9,U8),
      sumr(L4S1C2S3S4,U10,U9),
      sumr(L4S1S2C3S4,Y,U10).

calcul_valeur_Z(_,T2,T3,T4,Z) :-

/*     Z :=  L0 + L1 - L2S2 - L3S2C3 + L3C2S3 - L3S2C3C4
            + L4C2S3C4 + L4S2S3S4 - L4C2C3S4.                */

      cos(T2,C2),          sin(T2,S2),
      cos(T3,C3),          sin(T3,S3),
      cos(T4,C4),          sin(T4,S4),
      timesr(C2,C3,C2C3), timesr(C2,S3,C2S3),
      timesr(S2,C3,S2C3), timesr(S2,S3,S2S3),
      timesr(real(220000000,3),S2,L2S2),
      timesr(real(160000000,3),S2C3,L3S2C3),
      timesr(real(160000000,3),C2S3,L3C2S3),
      timesr(S2C3,C4,S2C3C4),
      timesr(real(160000000,3),S2C3C4,L3S2C3C4),
      timesr(C2S3,C4,C2S3C4),
      timesr(real(204600000,3),C2S3C4,L4C2S3C4),
      timesr(S2S3,S4,S2S3S4),
      timesr(real(204600000,3),S2S3S4,L4S2S3S4),
      timesr(C2C3,S4,C2C3S4),
      timesr(real(204600000,3),C2C3S4,L4C2C3S4),
      sumr(L2S2,U11,real(250000000,3)),
      sumr(L3S2C3,U12,U11),
      sumr(L3C2S3,U12,U13),
      sumr(L3S2C3C4,U14,U13),
      sumr(L4C2S3C4,U14,U15),
      sumr(L4S2S3S4,U15,U16),
      sumr(L4C2C3S4,Z,U16).

calcul_valeur_tangage(_,T2,T3,T4,T) :-
```

```
/*   T :-  T2 + T3 + T4        */

     sumr(T2,T3,U17),
     sumr(U17,T4,TT),
     signe_oppose(TT,T).


conversion_en_entier2(real(Mx,Ex),real(My,Ey),real(Mz,Ez),real(Mt,Et),X1,Y1,Z1,Tg):-
    Ex1 is Ex + 2,
    Ey1 is Ey + 2,
    Ez1 is Ez + 2,
    Et1 is Et + 2,
    int(real(Mx,Ex1),X1),
    int(real(My,Ey1),Y1),
    int(real(Mz,Ez1),Z1),
    int(real(Mt,Et1),Tg).

conversion_en_reel2(A,B,C,D,E,Ar,Br,Cr,Dr,Er):-
     cv_reel(A,Ar),
     cv_reel(B,Br),
     cv_reel(C,Cr),
     cv_reel(D,Dr),
     cv_reel(E,Er),!.

cv_reel2(0,real(0,0)) :- !.
cv_reel2(X,real(M,E)) :-
     int(real(M,E1),X),
     E is E1 - 2.
```

```
/*
MODULE DE LA CINEMATIQUE        methode "sur mesure"
-------------------------

Version plus intuitive et plus efficace
*/

transformation(coord(A1,A2,A3,A4,A5),tcp(X1,Y1,Z1,Tg,A5,A1)) :-
     conversion_en_reel2(A1,A2,A3,A4,A5,T1,T2,T3,T4,T5),
     sumr(T2,T3,U1),      sumr(U1,T4,U2),
     sin(T2,S1),          cos(T2,C1),
     sin(U1,S2),          cos(U1,C2),
     sin(U2,S3),          cos(U2,C3),
     sin(T1,S0),          cos(T1,C0),

/* Valeur de Z */
     timesr(S1,real(220000000,3),R1),
     timesr(S2,real(160000000,3),R2),
     timesr(S3,real(204600000,3),R3),
     sumr(R1,R2,R4),
     sumr(R3,R4,R5),
     sumr(R5,real(250000000,3),Z),

/* Valeur de X */
     timesr(C1,real(220000000,3),R6),
     timesr(C2,real(160000000,3),R7),
     timesr(C3,real(204600000,3),R8),
     sumr(R6,R7,R9),
     sumr(R8,R9,R10),
     timesr(C0,R10,X),

/* Valeur de Y */
     timesr(S0,R10,Y),

/* Valeur du tangage */
    sumr(T2,T3,R11),
    sumr(R11,T4,TT),
    signe_oppose(TT,T),
    conversion_en_entier2(X,Y,Z,T,X1,Y1,Z1,Tg),!.


conversion_en_entier2(real(Mx,Ex),real(My,Ey),real(Mz,Ez),real(Mt,Et),X1,Y1,Z1,Tg):-
    Ex1 is Ex + 2,
    Ey1 is Ey + 2,
    Ez1 is Ez + 2,
    Et1 is Et + 2,
    int(real(Mx,Ex1),X1),
    int(real(My,Ey1),Y1),
    int(real(Mz,Ez1),Z1),
    int(real(Mt,Et1),Tg),!.

conversion_en_reel2(A,B,C,D,E,Ar,Br,Cr,Dr,Er):-
     cv_reel(A,Ar),
     cv_reel(B,Br),
     cv_reel(C,Cr),
     cv_reel(D,Dr),
     cv_reel(E,Er),!.

cv_reel2(0,real(0,0)) :- !.
cv_reel2(X,real(M,E)) :-
     int(real(M,E1),X),
     E is E1 - 2.
```

```
/*
MODULE DE LA CINEMATIQUE INVERSE.
---------------------------------

     Le module de la cinematique inverse a pour effet de calculer les valeurs
     des angles de rotation des articulations d'une chaine mecanique articulee.
     Dans ce module, la theorie de la cinematique inverse est appliquee au
     robot   RM 501.
     L'interface du module ne comprend que fonction ' transformation_inverse '.
     Le module utilise l'extension floating point du Prolog.


*/

transformation_inverse(tcp(X,Y,Z,T,R,T1), coord(T1,T2,T3,T4,R)) :-
     var(T1),!,                                    /* Valeur par defaut du lacet */
     conversion_en_reel(X,Y,Z,T,Xr,Yr,Zr,Tr),
     resolution(Xr,Yr,Zr,Tr,Lr,Tr1,Tr2,Tr3,Tr4),
     conversion_en_entier(Tr1,Tr2,Tr3,Tr4,T1,T2,T3,T4),!,
     mecaniquement_compatible(T1,T2,T3,T4,R),!.

transformation_inverse(tcp(X,Y,Z,T,R,L), coord(T1,T2,T3,T4,R)) :-
     nonvar(L),
     conversion_en_reel(X,Y,Z,T,L,Xr,Yr,Zr,Tr,Lr),
     resolution(Xr,Yr,Zr,Tr,Lr,Tr1,Tr2,Tr3,Tr4),
     conversion_en_entier(Tr1,Tr2,Tr3,Tr4,T1,T2,T3,T4),
     mecaniquement_compatible(T1,T2,T3,T4,R),!.

conversion_en_reel(X,Y,Z,T,L,Xr,Yr,Zr,Tr,Lr):-
     cv_reel(X,Xr),
     cv_reel(Y,Yr),
     cv_reel(Z,Zr),
     cv_reel(T,Tr),
     cv_reel(L,Lr),!.

conversion_en_reel(X,Y,Z,T,Xr,Yr,Zr,Tr):-
     cv_reel(X,Xr),
     cv_reel(Y,Yr),
     cv_reel(Z,Zr),
     cv_reel(T,Tr),!.

cv_reel(0,real(0,0)) :- !.
cv_reel(X,real(M,E)) :-
     int(real(M,E1),X),
     E is E1 - 2.

conversion_en_entier(real(M1,E1),real(M2,E2),real(M3,E3),real(M4,E4),T1,T2,T3,T4):-
    Er1 is E1 + 2, int(real(M1,Er1),T1),
    Er2 is E2 + 2, int(real(M2,Er2),T2),
    Er3 is E3 + 2, int(real(M3,Er3),T3),
    Er4 is E4 + 2, int(real(M4,Er4),T4).

resolution(X,Y,Z,T,L,T1,T2,T3,T4) :-
     angle_corps(X,Y,T1),
     position_poignet(X,Y,Z,T,L,T1,PX1,PZ1),!,
     geometriquement_compatible(PX1,PZ1),
    angle_coude(PX1,PZ1,T3),
    angle_epaule(PX1,PZ1,T3,T2),
    angle_poignet(T2,T3,T,T4).
```

```
mecaniquement_compatible(A1,A2,A3,A4,A5) :-
     A1 >= 0,        A1 =< 30000,
     A2 >= ~1000,    A2 =< 12000,
     A3 >= ~9000,    A3 =< 0,
     A4 >= ~9000,    A4 =< 9000,
     A5 >= ~18000,   A5 =< 18000.

angle_corps(real(Mx,Ex),real(My,Ey),T) :-          /* premier quadrant */
     Mx > 0,
     My >= 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T,T1),!.

angle_corps(real(Mx,Ex),real(My,Ey),T) :-          /* deuxieme quadrant */
     Mx < 0,
     My >= 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(180000000,3),T2,T),!.

angle_corps(real(Mx,Ex),real(My,Ey),T) :-          /* troisieme quadrant */
     Mx < 0,
     My =< 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(180000000,3),T2,T),!.

angle_corps(real(Mx,Ex),real(My,Ey),T) :-          /* quatrieme quadrant */
     Mx > 0,
     My =< 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(360000000,3),T2,T),!.


angle_corps(real(0,0),real(0,0),T) :-
     actuelle(coord(T1,_,_,_,_)),
     cv_reel(T1,T),!.

angle_corps(real(0,0),real(M,_),real(900000000,2)) :-
     M >0,!.

angle_corps(real(0,0),real(M,_),real(270000000,3)) :-
     M < 0,!.

positicn_poignet(X,Y,Z,T,L,L,PX,PZ):-      /* Lacet = Angle du corps    */
     sqr(X,X1),
     sqr(Y,Y1),
     sumr(X1,Y1,Z1),
     sqr(Z2,Z1),
     cos(T,Z3),
     timesr(real(204600000, 3),Z3,Z4),
     sumr(Z4,PX,Z2),
     sin(T,Z6),
     timesr(real(299600000, 3),Z6,Z7),
     sumr(Z7,Z,Z8),
     sumr(Z8,real(~250000000, 3),PZ),!.
```

```
position_poignet(X,Y,Z,T,L,P,PX,PZ):-    /* Lacet = Angle_corps + 180 degres */
     sumr(P,real(180000000,3),L),
     sqr(X,X1),
     sqr(Y,Y1),
     sumr(X1,Y1,Z1),
     sqr(Z2,Z1),
     cos(T,Z3),
     timesr(real(299600000, 3),Z3,Z4),
     sumr(Z4,Z2,PX),
     sin(T,Z6),
     timesr(real(204600000, 3),Z6,Z7),
     sumr(Z7,Z,Z8),
     sumr(Z8,real(~250000000, 3),PZ),!.

position_poignet(X,Y,Z,T,L,P,PX,PZ):-    /* Lacet = Angle_corps - 180 degres */
     sumr(L, real(180000000,3),P),
     sqr(X,X1),
     sqr(Y,Y1),
     sumr(X1,Y1,Z1),
     sqr(Z2,Z1),
     cos(T,Z3),
     timesr(real(204600000, 3),Z3,Z4),
     sumr(Z4,Z2,PX),
     sin(TT,Z6),
     timesr(real(204600000, 3),Z6,Z7),
     sumr(Z7,Z,Z8),
     sumr(Z8,real(~250000000, 3),PZ).

geometriquement_compatible(PX1,PZ1) :-
     sqr(PX1,PX2),
     sqr(PZ1,PZ2),
     sumr(PX2,PZ2,D2),
     sqr(D1,D2),
     int(D1,D),
     D =< 380,
     D >= 272.

angle_coude(PX1,PZ1,real(M,E)) :-
     sqr(PX1,PX2),
     sqr(PZ1,PZ2),
     sumr(PX2,PZ2,D2),
     sumr(real(740000000,5),D1,D2),
     timesr(real(704000000,5),D3,D1),
     cos(real(M1,E),D3),
     M is ~M1.

angle_epaule(PX1,PZ1,T3,T2):-
     arctangeante(PX1,PZ1,P1),
     sin(T3,U1),
     timesr(real(160000000,3),U1.U2).
     cos(T3,U3),
     timesr(real(160000000,3),U3,U4),
     sumr(real(220000000,3),U4,U5),
     arctangeante(U5,U2,P2),
     sumr(T2,P2,P1).

angle_poignet(T2,T3,T,real(M,E)):-
     signe_oppose(T,TT),
     sumr(T2,T3,R),
     sumr(R,real(M,E),TT).

arctangeante(real(0,0),_,real(900000000,2)) :- !.
arctangeante(X,Y,A) :-
     timesr(X,P,Y),
     tan(A,P).
```

## AMELIORATION DES PERFORMANCES DU SYSTEME

Le pilotage du robot en mode cartésien nécessite l'utilisation très fréquente des modules de la cinématique et de la cinématique inverse.

Ces modules en TCP dure deux secondes environ en utilisant le deuxième module de la cinématique (en utilisant le premier, on en mettrait cinq).

La transformation d'un TCP en position dure 4 secondes. Ces performances médiocres sont dues d'une part au nombre très important d'unifications effectuées pendant le calcul et d'autre part au nombre très important de conversions et de normalisations. Certains environnements prolog mettent un compilateur (ou évaluateur partiel) à la disposition de l'utilisateur.

Si cela avait été le cas pour le York Prolog, j'aurais compilé les prédicats "transformation" et "transformation inverse". Comme ce n'est pas le cas et comme nous avons la possibilité de modifier l'interpréteur, j'ai implémenté deux nouveaux prédicats builtin "transf" et "transfi" qui effectuent respectivement la transformation d'une position en TCP et la transformation inverse.

Les modules de la cinématique et de la cinématique inverse sont reduits à peu de chose

### MODULE DE LA CINEMATIQUE

```
transformation(coord(A1,A2,A3,A4,A5),tcp(X,Y,Z,T,A5,A1)) :-
    transf(A1,A2,A3,A4,X,Y,Z),
    T is ~(A2 + A3 + A4).
```

### MODULE DE LA CINEMATIQUE INVERSE

```
transformation_inverse(tcp(X,Y,Z,T,R,T1), coord(T1,T2,T3,T4,R)) :-
    transfi(X,Y,Z,T,T1,T2,T3,T4),
    mecaniquement_compatible(T1,T2,T3,T4,R).

mecaniquement_compatible(A1,A2,A3,A4,A5) :-
    A1 >= 0,        A1 =< 30000,
    A2 >= ~1000,    A2 =< 12000,
    A3 >= ~9000,    A3 =< 0,
    A4 >= ~9000,    A4 =< 9000,
    A5 >= ~18000,   A5 =< 18000.
```

...

## Implenentation des prédicats Transf et Transfi

```
Procedure DoTransfi;
var X,Y,Z,T,T1,T2,T3,T4 : longint;
    R1,R2,R3,R4,R5 : boolean;
begin
    X := evaluate(argval[1],0);
    Y := evaluate(argval[2],0);
    Z := evaluate(argval[3],0);
    T := evaluate(argval[4],0);
    R1 := Resolution(X,Y,Z,T,T1,T2,T3,T4);
    R2 := IntResult(argval[5],callenv,T1);
    R3 := IntResult(argval[6],callenv,T2);
    R4 := IntResult(argval[7],callenv,T3);
    R5 := IntResult(argval[8],callenv,T4);
    result := R1 and R2 and R3 and R4 and R5
end;

Procedure DoPosTcp;
var X,Y,Z,T1,T2,T3,T4 : longint;
    R1,R2,R3 : boolean;
begin
    T1 := evaluate(argval[1],0);
    T2 := evaluate(argval[2],0);
    T3 := evaluate(argval[3],0);
    T4 := evaluate(argval[4],0);
    Position_Tcp(T1,T2,T3,T4,X,Y,Z);
    R1 := IntResult(argval[5],callenv,X);
    R2 := IntResult(argval[6],callenv,Y);
    R3 := IntResult(argval[7],callenv,Z);
   result := R1 and R2 and R3
end;
```

```
{ $S Trsf }
Unit Transf;

INTERFACE

Function Resolution(XI,YI,ZI,TI:longint;
                    Var Corps,Epaule,Coude,Poignet:longint):boolean;

Procedure Position_tcp( Corps,Epaule,Coude,Poignet:longint;Var X,Y,Z:longint);

IMPLEMENTATION

   const Pi     = 3.14159265;
         DeuxPi = 6.28318530;
         PiDemi = 1.57079632;
         L1 = 250;
         L2 = 220;
         L3 = 160;
         l4 = 207.5;
```

```
Function Resolution;        (* transformation  x,y,z ---> θ1 ... θ5 *)

var  Px,Py,X,Y,Z,T,A_Corps,A_Epaule,A_Coude,A_Poignet:real;
     I : integer;
     R,S,U : real;

begin
     X:= XI / 100;
     Y:= YI / 100;
     Z:= ZI / 100;
     T:= (TI / 18000) * Pi;
     if (Y = 0) and (X = 0)  then A_corps := 0                    else
     if (Y >= 0) and (X > 0) then A_corps := arctan(Y / X)        else
     if (Y > 0) and (X = 0)  then A_corps := PiDemi               else
     if (Y > 0) and (X < 0)  then A_corps := arctan(Y / X) + Pi   else
     if (Y <= 0) and (X < 0) then A_corps := arctan(Y / X) + Pi   else
     if (Y < 0) and (X = 0)  then A_corps := 3 * PiDemi           else
     if (Y < 0) and (X > 0)  then A_corps := arctan(Y / X) + DeuxPi ;
     Px := Sqrt(Sqr(X) + Sqr(Y)) - L4 * cos(T);
     Py := Z + L4  * sin(T) - L1;
     I := Trunc(Sqrt(Sqr(PX) + Sqr(PY)));
     if not (I <= 380) and (I >= 272) then begin
          Corps    := 36000;
          Epaule   := 36000;
          Coude    := 36000;
          Poignet  := 36000;
          Resolution := false
     end
     else begin
          R := (Sqr(PX) + Sqr(PY) - 74000) / 70400;
          if R <> 0 then A_coude := -arctan(sqrt( 1 - sqr(R)) / R)
                    else A_coude := PiDemi;
          if PX = 0 then R := PiDemi
                    else R := arctan(PY /PX);
          S := L3 * cos(A_Coude) + L2 ;
          if S = 0 then U := PiDemi
                   else U := arctan(L3 * sin(A_Coude) / S);
          A_Epaule := R - U;
          A_poignet:= -T - A_Coude - A_Epaule;
          Corps    := Round(A_Corps *18000 / Pi);
          Epaule   := Round(A_Epaule *18000 / Pi);
          Coude    := Round(A_Coude *18000 / Pi);
          Poignet  := Round(A_Poignet *18000 / Pi);
          Resolution := true;
     end
end;

Procedure Position_tcp;        (* transformation θ1 .... θ5 ---> x y z *)

Var Px,Py,Pz,A_Corps,A_Epaule,A_Coude,A_Poignet:real;

begin
     A_Corps    := (Corps / 18000)    * Pi;
     A_Epaule   := (Epaule / 18000)   * Pi;
     A_Coude    := (Coude / 18000)    * Pi;
     A_Poignet  := (Poignet / 18000)  * Pi;
     Px := cos(A_Epaule) * L2 + cos(A_Epaule + A_Coude) * L3 +
           cos(A_Epaule + A_Coude + A_Poignet) * L4 ;
     Py := sin(A_corps) * Px;
     Px := cos(A_corps) * Px;
     Pz := sin(A_Epaule) * L2  + sin(A_Epaule + A_Coude) * L3 +
           sin(A_Epaule + A_Coude + A_Poignet) * L4 ;
     X :=  Round(Px * 100);
     Y :=  Round(Py * 100);
     Z :=  Round((Pz + L1) * 100)
end;
end.
```

# ANNEXE D

## IMPLEMENTATION DE LA COUCHE XYZ

```
/*
IMPLEMENTATION DES COMPETENCES DES OBJETS DE LA COUCHE XYZ.
_____________________________________________________________

Competences du TCP
__________________

Transformation(Position_articulaire,Tcp)
----------------------------------------
    Voir module de la cinematique Annexe C .

Transformation_inverse(Position_articulaire,Tcp)
------------------------------------------------
    Voir module de la cinematique_inverse Annexe C

Lacet(Tcp,Angle)
----------------
*/
lacet(tcp(X,Y,_,_,_,L),L) :-
     cvrt_reel(X,XR),
     cvrt_reel(Y,YR),
     angle_lacet(XR,YR,LR),
     cvrt_entier(LR,L),!.

angle_lacet(real(Mx,Ex),real(My,Ey),T) :-          /* premier quadrant */
     Mx > 0,
     My >= 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T,T1),!.
angle_lacet(real(Mx,Ex),real(My,Ey),T) :-          /* deuxieme quadrant */
     Mx < 0,
     My >= 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(180000000,3),T2,T),!.
angle_lacet(real(Mx,Ex),real(My,Ey),T) :-          /* troisieme quadrant */
     Mx < 0,
     My =< 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(180000000,3),T2,T),!.
angle_lacet(real(Mx,Ex),real(My,Ey),T) :-          /* quatrieme quadrant */
     Mx > 0,
     My =< 0,
     timesr(T1,real(Mx,Ex),real(My,Ey)),
     tan(T2,T1),
     sumr(real(360000000,3),T2,T),!.
angle_lacet(real(0,0),real(0,0),T) :-
     actuelle(coord(T1,_,_,_,_)),
     cv_reel(T1,T),!.
angle_lacet(real(0,0),real(M,_),real(900000000,2)) :-
     M >0,!.
angle_lacet(real(0,0),real(M,_),real(270000000,3)) :-
     M < 0,!.
```

```
/*
Approche(Tcp1,Pente,Longueur,Tcp2)
----------------------------------
*/

approche(tcp(X,Y,Z,T,R,_),Pente,Lg,tcp(X1,Y1,Z1,T,R,_)):-
     transformation_inverse(tcp(X,Y,Z,T,R,_),coord(L,_,_,_,_)),
     conversion_en_reel(L,LR),
     conversion_en_reel(Pente,PR),
     conversion_en_reel(Lg,LGR),
     cos(PR,CPR),   cos(LR,CLR),
     sin(PR,SPR),   sin(LR,SLR),
     timesr(LGR,SPR,DZR),
     timesr(LGR,CPR,D1),
     timesr(D1,CLR,DXR),
     timesr(D1,SLR,DYR),
     conversion_en_entier(DXR,DX),
     conversion_en_entier(DYR,DY),
     conversion_en_entier(DZR,DZ),
     X1 is X - DX,
     Y1 is Y - DY,
     Z1 is Z + DZ,!.

*/
calcule_actuel(Tcp)
-------------------
*/

calcule_actuel(Tcp) :-
     actuelle(Pos),
     transformation(Pos,Tcp),
     modifier_tcp_actuel(Tcp),!.

/*
Initialisation_robot
--------------------
*/
initialisation_robot :-
     tcp_actuel(_),
     nest,
     calcule_actuel(Tcp),!.

initialisation_robot :-
     nest,
     reference(coord(0,~10250,9000,9000,1875)),
     calcule_actuel(Tcp),!.
/*
Tcp_actuel(Tcp)
---------------
*/

tcp_actuel(Tcp) :- actuel(Tcp).

*/
Competences du deplacement
--------------------------
```

```
immediat(Deplacement)
---------------------
*/

immediat(dep(Vit,Tcp)) :-
     transformation_inverse(Tcp,Pos),
     immediat(mvt(Vit,Pos)),
     retract(actuel(_)),
     assert(actuel(Tcp)),!.

/*
Vitesse_dep(Deplacement,Vitesse)
--------------------------------
*/

vitesse(dep(Vit,_),Vit).


/*
Destination_dep(Deplacement,Tcp)
--------------------------------
*/

destination(dep(_,Tcp),Tcp).


*/
Dernier_dep(Deplacement)
------------------------
*/
dernier_dep(dep(Vit,Tcp)) :-
     dernier(mvt(Vit,Pos)),
     actuel(Tcp).


Competences du tube
___________________

Tolerance(Tube,Tol)
-------------------
*/
tolerence(tube(tol(T),_,_),tol(T)).

/*
vitesse_tube(Tube,Vit)
----------------------
*/

vitesse(tube(_,vit(V),_),vit(V)).

destination(tube(_,_,tcp(X,Y,Z,T,R,L)),tcp(X,Y,Z,T,R,L)).

/*
destination_tube(Tube,Dest)
---------------------------
```

```
destination(tube(_,_,tcp(X,Y,Z,T,R,L)),tcp(X,Y,Z,T,R,L)).

/*
parcourir(Tube)
---------------
*/
parcourir(tube(Tol,Vit,Tcp)) :-
     actuel(Atcp),
     construire_trajectoire(Atcp,tube(Tol,Vit,Tcp),Traj),
     parcourir(Traj),
     modifier_tcp_actuel(Tcp),!.

/*
Competence du circuit
---------------------

destination_circ(Circuit,Tcp)
-----------------------------
*/

destination_circ(circ([tube(_,_,Tcp)]),Tcp).
destination_circ(circ([tube1|R]),Tcp):-
     destination_circ(circ(R),Tcp).

*/
Parcourir_circ(Circuit).
------------------------
*/

parcourir(circ([T1|T2])) :-
     actuel(Tcp),
     constr_lg_traj(Tcp,[T1|T2],Traj),
     parcourir(traj(Traj)).

/*
Predicats utilises pour la definition des competences des objets de la couche XYZ
-----------------------------------------------------------------------------------
*/

constr_lg_traj(Tcp,[],[]):- modifier_tcp_actuel(Tcp),!.
constr_lg_traj(Tcp,[T1|T2],Traj) :-
     construire_trajectoire(Tcp,T1,traj(Traj1)),
     destination(T1,Tcp1),
     constr_lg_traj(Tcp1,T2,Traj2),
     fusionner(Traj1,Traj2,Traj).


construire_trajectoire(Atcp,tube(Tol,Vit,Tcp),Traj) :-
     transformation_inverse(Tcp,Pos),
     ecart_tcp_successifs(Atcp,tube(Tol,Vit,Tcp),N,Delta),
     constr(Tcp,Vit,N,Delta,traj([mvt(Vit,Pos)]),Traj).

constr(Tcp,_,0,_,T,T) :- !.
constr(tcp(X,Y,Z,T,R,_),Vit,N,delta(Dx,Dy,Dz,Dt,Dr,_),traj(U),Traj) :-
     X1 is X - Dx,
     Y1 is Y - Dy,
     Z1 is Z - Dz,
     T1 is T - Dt,
     R1 is R - Dr,
     transformation_inverse(tcp(X1,Y1,Z1,T1,R1,_),Pos),
     N1 is N - 1,
     constr(tcp(X1,Y1,Z1,T1,R1,_),Vit,N1,delta(Dx,Dy,Dz,Dt,Dr,_),
                                          traj([mvt(Vit,Pos)|U]),Traj),!.
```

```
constr(tcp(X,Y,Z,T,R,_),Vit,N,delta(Dx,Dy,Dz,Dt,Dr,_),traj(U),Traj) :-
     X1 is X - Dx,
     Y1 is Y - Dy,
     Z1 is Z - Dz,
     T1 is T - Dt,
     R1 is R - Dr,
     N1 is N - 1,
     constr(tcp(X1,Y1,Z1,T1,R1,_),Vit,N1,delta(Dx,Dy,Dz,Dt,Dr,D1),traj(U),Traj).

ecart_tcp_successifs(tcp(Xa,Ya,Za,Ta,Ra,La),tube(Tol,Vit,tcp(X,Y,Z,T,R,L)),
                                                N1,delta(Dx,Dy,Dz,Dt,Dr,D1)) :-
     X1 is X - Xa,
     Y1 is Y - Ya,
     Z1 is Z - Za,
     T1 is T - Ta,
     R1 is R - Ra,
     int(Xr,X1),
     int(Yr,Y1),
     int(Zr,Z1),
     sqr(Xr,Sqrxr),
     sqr(Yr,Sqryr),
     sqr(Zr,Sqrzr),
     sumr(Sqrxr,Sqryr,Tr1),
     sumr(Tr1,Sqrzr,Tr2),
     sqr(Distr,Tr2),
     int(Distr,Dist),
     dist_max_entre_tcp(Tol,Dmax),
     N is (Dist / Dmax),
     N >0,
     Dx is X1 / N,
     Dy is Y1 / N,
     Dz is Z1 / N,
     Dt is T1 / N,
     Dr is R1 / N,
     N1 is N - 1,!.

ecart_tcp_successifs(_,tube(Tol,Vit,tcp(X,Y,Z,T,R,L)), 0,_).

dist_max_entre_tcp(tol(T),D) :- D is T * 50.

modifier_tcp_actuel(Tcp):-
     retract(actuel(_)),
     assert(actuel(Tcp)),!.

modifier_tcp_actuel(Tcp):-
     assert(actuel(Tcp)).
```

```
cvrt_reel(0,real(0,0)) :- !.
cvrt_reel(X,real(M,E)) :-
     int(real(M,E1),X),
     E is E1 - 2.

cvrt_entier(real(0,0),0) :- !.
cvrt_entier(real(M,E),X) :-
     E1 is E + 2,
     int(real(M,E1),X).

fusionner([], X, X).
fusionner([X|Y], U, [X|Z]) :-
          fusionner(Y, U, Z).

conversion_en_reel(0,real(0,0)) :-!.
conversion_en_reel(X,real(M,E)) :-
     int(real(M,E1),X),
     E is E1 - 2,!.

conversion_en_entier(real(M,E),X) :-
     E1 is E + 2,
     int(real(M,E1),X),!.


?- consult(fp2).
?- consult(cinematique_inv_2).
?- consult(cinematique3).
?- consult(interface_art).
```

# ANNEXE E

## REALISATION D'UN BOITIER D'APPRENTISSAGE

## (CODE PROLOG)

---

```
/*
MODULE D'APPRENTISSAGE

----------------------
 controle de l'ecran */

moveto(X,Y) :-
     X1 is X + 32,
     Y1 is Y + 32,
     Y1 >= 32, Y1 =< 63,
     X1 >= 32, X1 =< 119,
     put(27),put(61),put(Y1),put(X1).


effacer :- put(27),put(42).

writeto(X,Y,Z) :-
     moveto(X,Y),
     write(Z).

readfrom(X,Y,Z) :-
     writeto(X,Y,'      '),
     moveto(X,Y),
     read(Z),
     writeto(X,Y,'      '),
     writeto(X,Y,Z).

/* programme teacher */

teacher :-
     initialisation,
     repeat,
          lire_commande(X),
          traiter_commande(X),
     X == 1,!,
     terminaison.


/*  Initialisation */

initialisation:-
     effacer,
     writeto(0,2,'_____________________________________________'),
     writeto(44,2,'_____________________________________________'),
     writeto(30,5,'T E A C H I N G   B O X'),
     writeto(30,6,'_________________________'),
     writeto(0,10,'Positions cartesiennes       X ='),
     writeto(0,11,'                             Y ='),
     writeto(0,12,'                             Z ='),
     writeto(58,10,'Tangage =         degres'),
     writeto(58,12,'Roulis  =         degres'),
     writeto(0,14,'Positions articulaires   Corps ='),
     writeto(0,15,'                        Epaule ='),
     writeto(0,16,'                         Coude ='),
     writeto(0,17,'                    Inclinaison ='),
     writeto(0,18,'                        Torsion ='),
```

```
      writeto(58,14,'Vitesse =          mm/s'),
      writeto(58,16,'Pas     =          mm'),
      writeto(58,18,'Pas rot =          Pas'),
      writeto(10,24,'Commande :'),
      writeto(0,28,'________________________________________________'),
      writeto(44,28,'________________________________________________'),
       initialisation_robot,
       immediat(mvt(vit(40),coord(0,0,0,0,0))),
       retractall(pas(_)),
       retractall(vitesse(_)),
       retractall(pas_rotation(_)),
       retractall(instruction(_)),
       assert(pas(5000)),
       assert(vitesse_t(40)),
       assert(pas_rotation(1000)),
       actuelle_pos(coord(T1,T2,T3,T4,T5)),
       calcule_actuel(tcp(X,Y,Z,T,R,L)),
       assert(stop),
       ecrire_champs1(X),
       ecrire_champs2(Y),
       ecrire_champs3(Z),
       ecrire_champs4(T),
       ecrire_champs5(R),
       ecrire_champs6(T1),
       ecrire_champs7(T2),
       ecrire_champs8(T3),
       ecrire_champs9(T4),
       ecrire_champs10(T5),
       ecrire_champs11(40),
       ecrire_champs12(5000),
       ecrire_champs13(1000).

/* ecriture des champs de l'ecran */

ecrire_champs1(X) :- writeto(35,10,'      '),writeto(35,10,X).
ecrire_champs2(X) :- writeto(35,11,'      '),writeto(35,11,X).
ecrire_champs3(X) :- writeto(35,12,'      '),writeto(35,12,X).
ecrire_champs4(X) :- writeto(68,10,'      '),writeto(68,10,X).
ecrire_champs5(X) :- writeto(68,12,'      '),writeto(68,12,X).
ecrire_champs6(X) :- writeto(35,14,'      '),writeto(35,14,X).
ecrire_champs7(X) :- writeto(35,15,'      '),writeto(35,15,X).
ecrire_champs8(X) :- writeto(35,16,'      '),writeto(35,16,X).
ecrire_champs9(X) :- writeto(35,17,'      '),writeto(35,17,X).
ecrire_champs10(X) :- writeto(35,18,'      '),writeto(35,18,X).
ecrire_champs11(X) :- writeto(68,14,'      '),writeto(68,14,X).
ecrire_champs12(X) :- writeto(68,16,'      '),writeto(68,16,X).
ecrire_champs13(X) :- writeto(68,18,'      '),writeto(68,18,X).

/* lecture de la commande et des champs */
lire_commande(X) :- moveto(35,24),get0(X).
lire_champs12(X) :- readfrom(68,16,X).
lire_champs13(X) :- readfrom(68,18,X).

/* terminaison */
terminaison :- nest,effacer.
```

```
/* traitement de commandes */

traiter_commande(1).                          /* fin de travail */
traiter_commande(2):- trace,!.
traiter_commande(3):- notrace,!.

traiter_commande(100) :-                      /* X := X + pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     X1 is X + P,
     immediat(dep(vit(V),tcp(X1,Y,Z,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs1(X1),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(104) :-                            /* X := X - Pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     X1 is X - P,
     immediat(dep(vit(V),tcp(X1,Y,Z,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs1(X1),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(116) :-                            /* Y := Y + Pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     Y1 is Y + P,
     immediat(dep(vit(V),tcp(X,Y1,Z,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs2(Y1),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(118) :-                                /* Y := Y - Pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     Y1 is Y - P,
     immediat(dep(vit(V),tcp(X,Y1,Z,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs2(Y1),
```

```
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(102) :-                              /* Z := Z + Pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     Z1 is Z + P,
     immediat(dep(vit(V),tcp(X,Y,Z1,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs3(Z1),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(103) :-                              /* Z := Z - Pas */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     vitesse_t(V),
     pas(P),
     Z1 is Z - P,
     immediat(dep(vit(V),tcp(X,Y,Z1,T,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs3(Z1),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(44) :-      /* Mise a jour du pas de rotation des articulations */
    lire_champs13(Z),
    integer(Z),
    retract(pas_rotation(_)),
    assert(pas_rotation(Z)),!.

traiter_commande(44) :-
    pas_rotation(X),
    ecrire_champs13(X),
    put(7),!.

traiter_commande(107) :-              /* Mise a jour du Pas */
    lire_champs12(Z),
    integer(Z),
    retract(pas(_)),
    assert(pas(Z)),!.

traiter_commande(107) :-
    pas(X),
    ecrire_champs12(X1),
    put(7),!.
```

```
traiter_commande(105) :-                /* Vitesse ;= Vitesse + 40 mm/s */
    vitesse_t(X),
    X1 is X + 40,
    X1 =< 400,!,
    retract(vitesse_t(_)),
    ecrire_champs11(X1),
    assert(vitesse_t(X1)),!.

traiter_commande(111) :-                /* Vitesse ;= Vitesse  - 40 mm/s */
    vitesse_t(X),
    X1 is X - 40,
    X1 =< 400,!,
    retract(vitesse_t(_)),
    ecrire_champs11(X1),
    assert(vitesse_t(X1)),!.

traiter_commande(121) :-             /* Tangage := Tangage + pas rotation */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     pas_rotation(P),
     TT is T + P,
     vitesse_t(V),
     immediat(dep(vit(V),tcp(X,Y,Z,TT,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs4(TT),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(110) :-             /* Tangage := Tangage - pas rotation */
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     pas_rotation(P),
     TT is T - P,
     vitesse_t(V),
     immediat(dep(vit(V),tcp(X,Y,Z,TT,R,_))),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     ecrire_champs4(TT),
     ecrire_champs6(T1),
     ecrire_champs7(T2),
     ecrire_champs8(T3),
     ecrire_champs9(T4),
     ecrire_champs10(T5),!.

traiter_commande(117) :-                /* Roulis := Roulis + pas rotation */
     pas_rotation(P),
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
     R1 is R + P,
     vitesse_t(V),
     immediat(dep(vit(V),tcp(X,Y,Z,T,R1,_))),
     ecrire_champs5(R1),
     ecrire_champs10(R1),!.

traiter_commande(109) :-                /* Roulis := Roulis - pas rotation */
     pas_rotation(P),
     tcp_actuel(tcp(X,Y,Z,T,R,_)),
```

```
     R1 is R - P,
     vitesse_t(V),
     immediat(dep(vit(V),tcp(X,Y,Z,T,R1,_))),
     ecrire_champs5(R1),
     ecrire_champs10(R1),!.

traiter_commande(32) :-       /* Enregistrement de la position actuelle du SMA */
     retractall(stop),
     vitesse_t(V),
     actuelle_pos(Pos),
     assertz(instruction(mvt(vit(V),Pos))),!.

traiter_commande(57) :- stop,!.

traiter_commande(57) :-              /* Interruption du mouvement continu */
     assert(stop),
     assertz(instruction(stop)),!.

traiter_commande(45) :-                  /* Fermeture de la pince */
     assert(stop),
     assertz(instruction(stop)),
     fermer(force(5,0,10)),
     assertz(instruction(close_grip)),!.

traiter_commande(61) :-                  /* Ouverture de la pince */
     assert(stop),
     assertz(instruction(stop)),
     ouvrir(force(5,0,10)),
     assertz(instruction(open_grip)),!.

traiter_commande(42) :-           /* Execution du programme enregistre */
     initialiser,
     retractall(trajectoire(_)),
     assert(trajectoire([])),
     assertz(instruction(fin_travail)),
     repeat,
         instruction(X),
         traiter(X),
     X == fin_travail,
     retract(instruction(fin_travail)),!.

     traiter(fin_travail) :-  trajectoire([]),!.

     traiter(fin_travail) :-
          retract(trajectoire(X)),
          parcourir(traj(X)),
          assert(trajectoire([])),!.

     traiter(stop) :- trajectoire([]),!.

     traiter(stop) :-
          retract(trajectoire(X)),
          parcourir(traj(X)),
          assert(trajectoire([])),!.

     traiter(open_grip) :-
```

```
        ouvrir(force(5,0,10)),!.

    traiter(close_grip) :-
        fermer(force(5,0,10)),!.

    traiter(mvt(X,Y)) :-
        retract(trajectoire(T)),
        append3(T,[mvt(X,Y)],U),
        assert(trajectoire(U)),!.

traiter_commande(53) :-            /*   Destruction du programme en memoire */
    retractall(instruction(_)),!.

traiter_commande(54) :-            /*     Sauvetage du programme enregitre  */
    tell('programme_robot'),
    save_all(instruction(X)),
    told,!.

    save_all(X) :-
        call(X),
        record(X),
        fail.
    save_all(X).

    record(X) :- write(X),write('.'),nl.

traiter_commande(55) :-       /* chargement d'un programme robot */
    see('programme_robot'),
    retractall(instruction(_)),
    repeat,
        read(X),
        trt(X),
    eof,
    seen,!.

    trt((?- end)).
    trt(X) :-assertz(X).

traiter_commande(49) :-         /* Corps := corps + pas    */
    pas_rotation(P),
    actuelle_pos(coord(T1,T2,T3,T4,T5)),
    R1 is T1 + P,
    valide(coord(R1,T2,T3,T4,T5)),!,
    vitesse_t(V),
    immediat(mvt(vit(V),coord(R1,T2,T3,T4,T5))),
    calcule_actuel(tcp(X,Y,Z,_,_,_)),
    ecrire_champs6(R1),
    ecrire_champs1(X),
    ecrire_champs2(Y),
    ecrire_champs3(Z),!.

traiter_commande(50) :-            /* Corps := corps - pas */
    pas_rotation(P),
    actuelle_pos(coord(T1,T2,T3,T4,T5)),
    R1 is T1 - P,
    valide(coord(R1,T2,T3,T4,T5)),!,
```

```
     vitesse_t(V),
     immediat(mvt(vit(V),coord(R1,T2,T3,T4,T5))),
     calcule_actuel(tcp(X,Y,Z,_,_,_)),
     ecrire_champs6(R1),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),!.

traiter_commande(113) :-                /* Epaule := epaule + pas */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R2 is T2 + P,
     valide(coord(T1,R2,T3,T4,T5)),!,
     vitesse_t(V),
     immediat(mvt(vit(V),coord(T1,R2,T3,T4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs7(R2),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(119) :-                /* Epaule := epaule - pas */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R2 is T2 - P,
     valide(coord(T1,R2,T3,T4,T5)),!,
     vitesse_t(V),
     immediat(mvt(vit(V),coord(T1,R2,T3,T4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs7(R2),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(97) :-                          /* Coude := coude + pas  */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R3 is T3 + P,
     valide(coord(T1,T2,R3,T4,T5)),!,
     vitesse_t(V),
     immediat(mvt(vit(V),coord(T1,T2,R3,T4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs8(R3),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(115) :-                        /* Coude := coude - pas */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R3 is T3 - P,
     valide(coord(T1,T2,R3,T4,T5)),!,
     vitesse_t(V),
```

```
     immediat(mvt(vit(V),coord(T1,T2,R3,T4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs8(R3),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(122) :-        /* Tangage poignet := tangage poignet  + pas */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R4 is T4 + P,
     valide(coord(T1,T2,T3,R4,T5)),!,
     vitesse_t(V),
     immediat(mvt(vit(V),coord(T1,T2,T3,R4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs9(R4),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(120) :-        /* Tangage poignet := tangage poignet  - pas */
     pas_rotation(P),
     actuelle_pos(coord(T1,T2,T3,T4,T5)),
     R4 is T4 - P,
     valide(coord(T1,T2,T3,R4,T5)),!,
     vitesse_t(V),
     immediat(mvt(vit(V),coord(T1,T2,T3,R4,T5))),
     calcule_actuel(tcp(X,Y,Z,T,_,_)),
     ecrire_champs9(R4),
     ecrire_champs1(X),
     ecrire_champs2(Y),
     ecrire_champs3(Z),
     ecrire_champs4(T),!.

traiter_commande(X) :- put(7).

append3([], X, X):- !.
append3([X|Y], U, [X|Z]) :-
          append3(Y, U, Z).


?- consult(interface_xyz).
```